5-2001

Knowledge-enhanced latent semantic indexing (KELSI): algorithms and applications

David Guo

To the Graduate Council:

I am submitting herewith a thesis written by David Guo entitled "Knowledge-enhanced latent semantic indexing (KELSI): algorithms and applications." I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.

Michael W. Berry, Major Professor

We have read this thesis and recommend its acceptance:

David Straight, Peiling Wang

Accepted for the Council:
Carolyn R. Hodges

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

To the Graduate Council:

I am submitting herewith a thesis written by David Guo entitled "Knowledge-Enhanced Latent Semantic Indexing (KELSI): Algorithms and Applications". I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.

Michael W. Berry

Dr. Michael W. Berry, Major Professor

We have read this thesis
and recommend its acceptance:

Accepted for the Council:

Interim Vice Provost and
Dean of the Graduate School

Knowledge-Enhanced Latent Semantic Indexing (KELSI): Algorithms and Applications

A Thesis

Presented for the

Master of Science Degree

The University of Tennessee, Knoxville

David Guo

May 2001

Acknowledgments

I thank my advisor, Dr. Michael Berry, for his tireless support and advice throughout this project. I would also like to thank Drs. David Straight and Peiling Wang for serving on my thesis committee.

I thank Dian Martin for going out of her way in helping me with LSI software environments and tools. I thank Bryan Thompson of Global Wisdom, Inc. for his ideas of adding concepts to LSI (LSI+C) and for pointing out the possibility of using MeSH headings. Finally, I thank Dr. Sidney Bailin of Knowledge Evolution, Inc. for helpful discussions on adding semantic structures into LSI.

Abstract

Latent Semantic Indexing (LSI) is a popular information retrieval model for concept-based searching. As with many vector space IR models, LSI requires an existing term-document association structure such as a term-by-document matrix. The term-by-document matrix, constructed during document parsing, can only capture weighted vocabulary occurrence patterns in the documents. However, for many knowledge domains (e.g., medicine) there are pre-existing semantic structures that could be used to organize and to categorize information. The goals of this study are to demonstrate how such semantic structures can be incorporated into the LSI vector space model and to measure their overall effect on query matching performance. The new approach, called Knowledge-Enhanced LSI (KELSI), is applied to documents in the OHSUMED medical abstracts using the semantic structures provided by the UMLS Semantic Network and MeSH. Results based on precision-recall graphs and 11-point average precision values (P) indicate that a MeSH-enhanced search index is capable of delivering noticeable incremental performance gain over the original LSI model – 28% improvement for P=.01 and 100% improvement for P=.30. This performance gain is achieved by replacing the original query with the MeSH heading extracted from the query text via regular expression matchs.

Contents

1 Introduction **1**

2 Building KELSI Search Indices **4**

2.1 LSI Overview 4

2.2 Overview of OHSUMED, UMLS, and MeSH 8

2.2.1 OHSUMED Collection 8

2.2.2 UMLS Semantic Network 8

2.2.3 MeSH Headings 9

2.3 General Approach for Adding Semantic Structures 10

2.4 Adding UMLS Concept Vectors 11

2.4.1 Mapping dictionary terms to UMLS concepts 11

2.4.2 Mapping the accepted concepts into UMLS Semantic Network tree nodes 13

2.4.3 Add Semantic Network nodes to term-by-document matrix 13

2.5 Adding MeSH Vectors 15

3 **Query-Matching Methods for KELSI** 16

3.1 KELSI with UMLS Semantic Network 19

3.1.1 Query Replacement 19

3.1.2 Adding Concept Vectors 21

3.1.3 Adding Projections 22

3.2 KELSI with MeSH Headings 23

3.2.1 Direct Query Match 26

3.2.2 Two-step Approach 27

3.2.3 Regular Expression Match 27

3.3 Performance Comparison 30

4 **Incremental Performance Gain** 32

4.1 Magnitude of Improvement 33

4.2 Incremental Performance Gain 33

5 **Summary and Conclusions** 41

Bibliography 45

Appendices 48

A **Sample Entries from OHSUMED and MeSH** 49

B **Results of Query Matching Using UMLS Concept Vectors** 54

C Results of Query Matching Using MeSH Vectors 60

Vita 66

List of Tables

3.1 Average cosine: relevant document versus centroid 18

3.2 MeSH regular expression match for query 37 28

3.3 A MeSH tree example 30

3.4 Performance comparison 31

4.1 Magnitude of performance improvement 34

4.2 Ranked return set for query 15 35

4.3 Ranked return sets for query 95 35

4.4 Incremental performance gain 36

4.5 Incremental performance gain for LSI_{UMLS} 39

4.6 Incremental performance gain for LSI_{MeSH} 40

A.1 A sample entry from the OHSUMED collection. 50

A.2 Data fields of an OHSUMED entry. 51

A.3 A sample entry from the MeSH heading description file. 52

A.4 A list of all top-level MeSH trees with the node *Chemicals and Drugs* expanded to the next level. 53

B.1 $\vec{cv}$ query replacement versus original LSI: P values. 56

B.2 $P_{q_{\vec{add}}}$ versus P_{LSI}. 58

B.3 $P_{q_{\vec{proj}}}$ versus P_{LSI}. 59

C.1 $P_{MeSH\ Direct\ Match}$ versus P_{LSI} for 106 queries. 62

C.2 $P_{MeSH\ Two-Step\ Match}$ versus P_{LSI} for 106 queries. 64

C.3 $P_{MeSH\ Regular\ Expression\ Match}$ versus P_{LSI} for 106 queries. 65

List of Figures

2.1 LSI flow chart 7

2.2 Distribution of concepts among the UMLS Semantic Network tree nodes. 14

3.1 $\vec{q_{proj}}$ versus original LSI : precision-recall graphs 24

3.2 MeSH vectors nearest to centroids versus LSI: precision-recall graphs 25

3.3 Regular expression matched MeSH vectors versus LSI: precision-recall graphs 29

B.1 Precision-recall graphs: $\vec{cv}$ query replacement versus original LSI. 55

B.2 Precision-recall graphs : $\vec{q_{add}}$ versus original LSI. 57

C.1 Precision-recall graphs: Direct-matched MeSH vectors versus LSI. 61

C.2 Precision-recall graphs: Two-step matched MeSH vectors versus LSI. 63

Chapter 1

Introduction

Latent Semantic Indexing (LSI) is a popular information retrieval model for concept-based searching [DDF$^+$90]. As with many vector space IR models, LSI requires an existing term-document association structure such as a term-by-document matrix [BB99]. The term-by-document matrix, constructed during document parsing, can only capture the (weighted) vocabulary occurrence patterns within the documents. However, for many knowledge domains (e.g., medicine) there are pre-existing semantic structures that could be used to organize and to categorize information. The goals of this study are to demonstrate how such semantic structures can be incorporated into the LSI vector space model and to measure their overall effect on query matching performance.

What motivated this study is the observation that if these semantic structures can be used to organize and to categorize information for hand search by humans, perhaps they can be similarly used for automated machine search encapsulated in an existing vector space model. In that case, components of the added semantic structures can be used to guide the query vector toward more relevant documents or be used to replace the query vector all together.

The approach presented in this thesis is called Knowledge-Enhanced LSI (KELSI). There are two key differences between KELSI and the original LSI method:

1. The original term-by-document matrix is augmented with additional concept-based vectors constructed from the semantic structures.

2. A number of query modification and query replacement methods (that exploit the semantic structures) are applied during query-matching.

Other than those two differences, the original LSI model remains intact. In fact, several of the analyses performed in this study utilize existing LSI software environments and tools [HTBM00] [LB97].

The document collection used for KELSI development is from the field of medical informatics – the Oregon Health Sciences University MEDLINE abstracts (OHSUMED) [HBLH94]. In addition, two medical semantic structures are used. They are the Unified Medical Language System (UMLS) [SHB97] Semantic Network and the Medical Subject Heading (MeSH) [NBB+00]. Each of those semantic structures is applied separately to produce two different enhanced search indices.

This thesis is organized as follows: Chapter 2 reviews how LSI and the new

KELSI vector space models are constructed. Chapter 3 discusses query-matching methods that exploit these enhanced vector space models. Chapter 4 evaluates the incremental performance gain from KELSI versus the original LSI method, and a summary with concluding remarks is provided in Chapter 5.

Chapter 2

Building KELSI Search Indices

This chapter begins with a brief overview of the LSI vector space model. The OHSUMED collection and two semantic structures (UMLS Semantic Network and MeSH headings) are introduced, and methods for building KELSI search indices are discussed.

2.1 LSI Overview

KELSI search indices are built by incorporating semantic structures (UMLS Semantic Network or MeSH headings) into the original LSI vector space model. Before discussing how KELSI search indices are constructed, it may be helpful to briefly review how the original LSI vector space model is constructed. First, a term-by-document matrix $A = [a_{ij}]$ is generated by parsing the document collection. Each matrix entry a_{ij} is a weighted representation of the occurrence of a word token within a document. For example,

$$a_{ij} = l_{ij} g_i d_j, \tag{2.1}$$

where l_{ij} is the local weight for the term i in document j, g_i is the global weight for the term i in the collection, and d_j is a document normalization factor which specifies whether or not the columns of A (i.e., the documents) are normalized [BB99]. The matrix A can be large and rather sparse[1]. The text parser also creates a dictionary of word tokens and their corresponding global weights. For this study, logarithmic local and global entropy weightings are used. For a detailed discussion of different term weighting schemes, see [BB99]. To model the latent structure of term-document associations represented by the term-by-document matrix A, a reduced-rank approximation to the matrix A is computed via the truncated Singular Value Decomposition (SVD) [GL96]. The optimal dimension of the reduced-rank approximation to A is still an open research question. For this study, 100 dimensions are used. Document parsing and SVD computations are implemented using the General Text Parser (GTP) software environment [HTBM00].

The SVD of the original term-by-document matrix can be written as [GL96]

$$A = U \Sigma V^T, \tag{2.2}$$

[1]Relatively few nonzero elements compared to zero elements.

where A is the $m \times n$ term-by-document matrix, U is an $m \times m$ orthogonal matrix whose columns define the left singular vectors of A; V is an $n \times n$ orthogonal matrix whose columns define the right singular vectors of A; and Σ is the $m \times n$ diagonal matrix containing the nonnegative singular values $\sigma_1 \geq \sigma_1 \geq \ldots \geq \sigma_{min(m,n)}$ of A in descending order along its diagonal. The k-dimensional reduced-rank approximation of A, denoted by A_k, is constructed by setting all but the k-largest singular values of A equal to zero so that

$$A_k = U_k \Sigma_k V_k^T, \tag{2.3}$$

where U_k and V_k comprise the first k columns of U and V, and Σ_k contains the k-largest singular values of A. Using the components of A_k, all terms and documents can be encoded as vectors in the k-dimensional space. For example, the j-th term and document vectors can be encoded as $t_j = \Sigma_k U_k^T e_j$ and $d_j = \Sigma_k V_k^T e_j$, respectively, where e_j denotes the j-th canonical vector of dimension n.

Query processing is done by first transforming a query into a *pseudo document* [BB99]. Given q, the vector whose non-zero elements correspond to the term weights (see Equation 2.2) of all valid query words, the *pseudo document* $\hat{q}$ can be represented by

$$\hat{q} = q^T U_k \Sigma_k^{-1}. \tag{2.4}$$

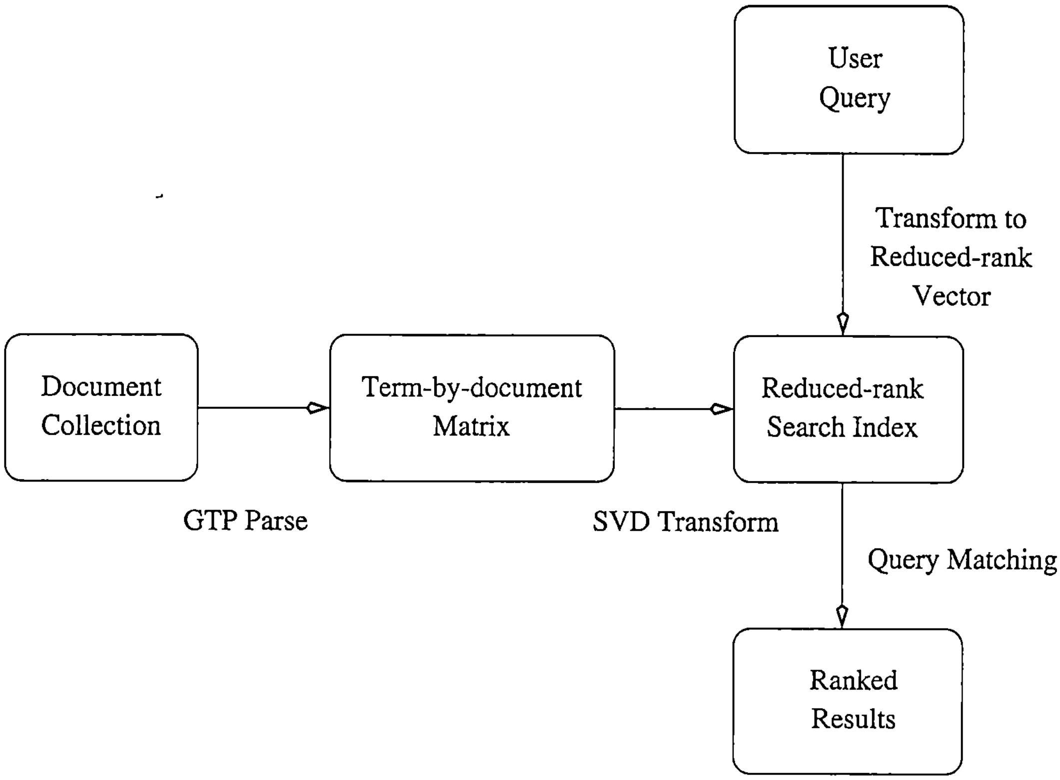

Figure 2.1: LSI flow chart

Thus, $\hat{q}$ is a k-dimensional vector spanned by A_k. This vector is then compared (via cosine calculations) with document or term encodings (also k-dimensional vectors) to generate ranked lists of similar documents or terms. Figure 2.1 is an illustration of the LSI query-matching process.

Semantic structures (UMLS Semantic Network or MeSH headings) can be added as either rows or columns into the original term-by-document matrix – whichever is more convenient. In either case, the newly added rows or columns are transformed to vectors in the reduced-rank space.

2.2 Overview of OHSUMED, UMLS, and MeSH

2.2.1 OHSUMED Collection

The OHSUMED collection was created to assist medical information retrieval. It contains 348,566 abstracts from 270 medical journals dating from 1987 to 1991 [HBLH94]. A sample entry of the OHSUMED collection appears in Appendix A.1.

In addition to the abstracts, OHSUMED also provides 106 test queries. Associated with each query is a list of documents that are judged to be relevant. Those relevance judgments are used to evaluate KELSI search performance.

2.2.2 UMLS Semantic Network

UMLS is a system of knowledge sources currently under development by the National Library of Medicine [SHB97]. It has three main components: the Semantic Network, the Metathesaurus, and the SPECIALIST Lexicon.

The UMLS Semantic Network is one of the semantic structures that this study will consider. It contains 134 nodes, where each node represents a knowledge category in the medical domain. The nodes are organized into a tree structure, and each node is populated by a number of concepts. A concept (comprised of one or more word tokens) is the basic unit of knowledge in UMLS. There are approximately 730,000 concepts in UMLS. The mapping between a concept and its constituent word tokens is defined by an ASCII relational table – MRXNS.ENG – in the Metathesaurus. Another Metathesaurus table – MRSTY – maps each concept into a Semantic Network tree node.

2.2.3 MeSH Headings

MeSH is the other semantic structure for consideration. It is a controlled vocabulary created by the National Library of Medicine [NBB+00]. It contains 19,942 headings and is used for indexing and cataloging articles and books related to medicine. Each MeSH heading has a description file. The *ENTRY* field of the description file tracks synonyms and alternate spellings, and allows MeSH to function as a thesaurus. A sample MeSH description entry appears in Appendix A.3.

In addition to providing descriptions, MeSH also defines the hierarchical relationships between headings. MeSH has 15 top level trees (Appendix A.4) and MeSH headings are assigned to those trees. Such MeSH headings, which are assigned by experienced human indexers, are specified in the .M fields of the entries in the OHSUMED collection.

Having introduced the document collection and the two pre-existing semantic structures (UMLS and MeSH), the subsequent sections discuss how to use those structures are used to construct the KELSI search indices.

2.3 General Approach for Adding Semantic Structures

The semantic structures generated from the UMLS Semantic Network are added as columns to the original term-by-document matrix A. As mentioned earlier, the Semantic Network is based on UMLS concepts. As these concepts may or may not be represented by the given document collection, their presence must be inferred from terms contained in the dictionary. In this case, the number of term vectors remain the same while concept vectors are added as columns (documents). If C denotes the new concept vectors constructed from the UMLS Semantic Network, then the augmented term-by-document matrix A_{UMLS} can be expressed as

$$A_{UMLS} = (A|C). \tag{2.5}$$

In comparison, the semantic structures related to MeSH headings are added as rows (or terms) to the original term-by-document matrix A. The MeSH headings are defined in the .M fields of the document collection, and they are extracted (during document parsing) as *special* tokens or phrases for the term-by-document matrix. If M denotes the row vectors associated with all parsed MeSH headings, then the augmented term-by-document matrix A_{MeSH} can be expressed as

$$A_{MeSH} = \left(\frac{A}{M}\right). \tag{2.6}$$

Having seen how the original LSI vector space model can be constructed and modified by the addition of new semantic structures, the next section discusses the inclusion of UMLS concepts and MeSH headings in more detail.

2.4 Adding UMLS Concept Vectors

Adding Semantic Network tree nodes to the original term-by-document matrix involves the following steps:

1. Map OHSUMED dictionary terms (created by GTP) to UMLS concepts.

2. Map those concepts into UMLS Semantic Network tree nodes.

3. Add Semantic Network nodes to the term-by-document matrix.

2.4.1 Mapping dictionary terms to UMLS concepts

Mapping OHSUMED dictionary terms to UMLS concepts requires two steps.

1. The entries of MRXNS.ENG are used to build a *word-to-concept* hash table. MRXNS.ENG is one of the ASCII relational tables in the Metathesaurus. Its key field is called the *Concept Unique Identifier* (CUI). MRXNS.ENG relates each CUI to a set of constituent word tokens. A sample entry of MRXNS.ENG is

   `ENG|compound drug iron poison|C0412842|L0793174|S0992160|.`

Here, `ENG` stands for English language entries; `compound drug iron poison` are the constituent word tokens for this concept; `C0412842` is the CUI for this concept, and the remaining two fields are called *Term Unique Identifier* and *String Unique Identifier*, respectively. The key field of the *word-to-concept* hash table entry is a word token and the value field contains all related CUI's and word counts for each CUI.

2. For each OHSUMED dictionary word, the corresponding entry is selected from the *word-to-concept* hash table that is keyed on the word. The value field is then used to build a new hash table called *holding.* This new hash table's key field is the CUI, and its value field contains the number of terms in the CUI and the number of hits on those terms from the dictionary. When the hit-count matches the number of terms in the CUI, that CUI is placed into the *accept* file. When the dictionary is exhausted, the entries remaining in the *holding* area can be accepted or rejected based on a user-specified threshold. For this study, a concept is accepted if all of its constituent word tokens appear in the dictionary. The number of matched concepts can be certainly increased by relaxing this constraint.

2.4.2 Mapping the accepted concepts into UMLS Semantic Network tree nodes

The UMLS Semantic Network defines a tree structure with 134 nodes. UMLS concepts are mapped into tree nodes through the Metathesaurus table MRSTY. A sample entry of MRSTY is

```
C0029122|T116|Amino Acid, Peptide, or Protein|.
```

Here, `C0029122` is the CUI, `T116` is the UMLS Semantic Network tree node identifier, and `Amino Acid, Peptide, or Protein` is the tree node name.

Using the acceptance threshold mentioned earlier, 238,160 concepts can be extracted from the document collection. To investigate properties of the Semantic Network, all accepted concepts are mapped into the tree nodes. The distribution of concepts among the tree nodes can vary a great deal (Figure 2.2). For example, the node *professional society* contains 14 concepts, whereas *disease or syndrome* contains 22,143 concepts. MRXNS.ENG reveals that a concept has on average five terms. When the concepts within each node are expanded to individual terms, the resulting concept vectors can be rather dense (spanning many dictionary terms).

2.4.3 Add Semantic Network nodes to term-by-document matrix

Having mapped the *accepted* concepts into UMLS Semantic Network tree nodes, the concepts in each node are expanded back into word tokens – using a *concept-to-*

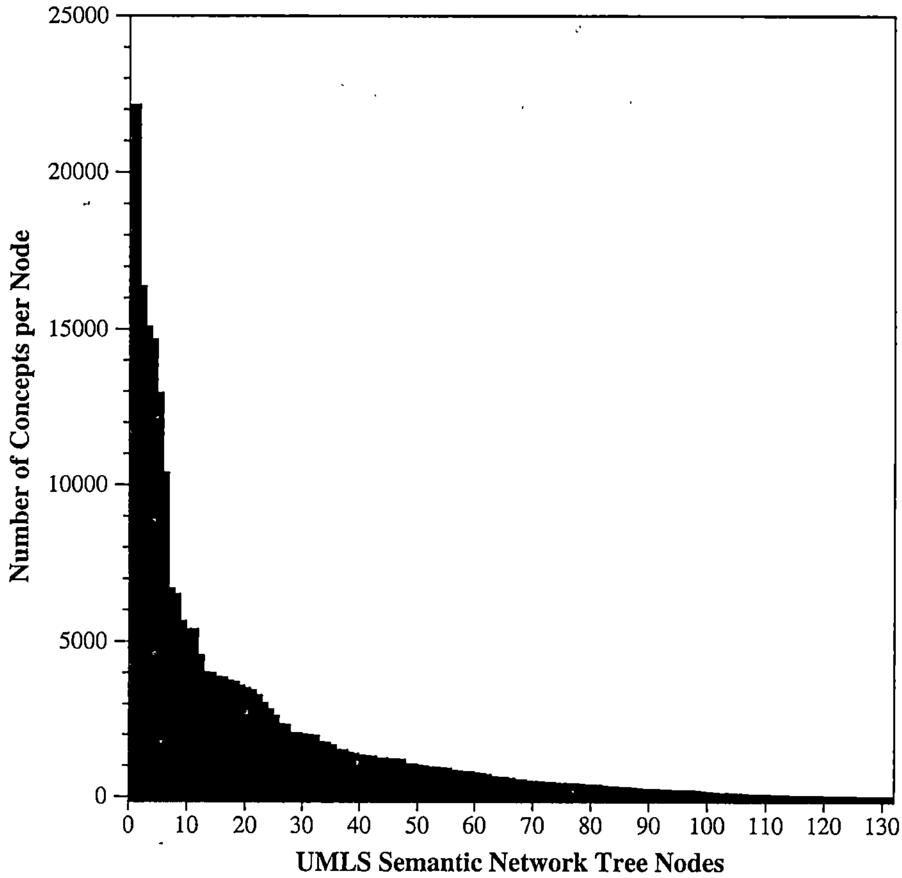

Figure 2.2: Distribution of concepts among the UMLS Semantic Network tree nodes.

word hash table constructed from MRXNS.ENG. At this point, UMLS Semantic Network nodes (each comprised by a set of word tokens) can be appended to the term-by-document matrix as columns vectors. The non-zero elements of these vectors (referred to as *UMLS concept vectors*) are simply the global weights for each word token in the dictionary. A reduced-rank approximation to the augmented term-by-document matrix (Equation 2.5) is then computed using the truncated

SVD.

2.5 Adding MeSH Vectors

Compared to the UMLS concept vectors, it is relatively simple to add MeSH headings into the original term-by-document matrix. Since the .M fields (Appendix A, Table A.1) of the OHSUMED documents contain the MeSH headings assigned to a document, they can be extracted during parsing using a special filter. With this new filter, 13,853 out of a total of 19,942 possible MeSH headings can be extracted from the OHSUMED collection. The words in each heading are concatenated into a single string, that can be added to the dictionary. A new row vector is then generated for the term-by-document matrix. During all subsequent encounters, MeSH headings are similarly concatenated to ensure consistency. The newly added row vectors will be referred to as *MeSH vectors.* The augmented term-by-document matrix (Equation 2.6) is then decomposed for the reduced-rank vector space model using the truncated SVD.

Chapter 3

Query-Matching Methods for KELSI

After the UMLS concept vectors and MeSH headings are incorporated into an LSI model, the question becomes how can they be used to enhance query performance?

There are two possibilities:

1. Replace the original query vectors generated by the LSI model with the newly added vectors, i.e., using them as *proxies*.

2. Modify the original LSI query vectors by aligning (or projecting) them toward the newly added vectors, i.e., using them as *guides*.

To see whether the new vectors can serve as proxies, their proximity to the relevant documents within the reduced rank space needs to be investigated. For each query, the OHSUMED collection provides a set of relevant documents. The centroid vector for each set of relevant documents is calculated using document vectors encoded (via the SVD) for 100-dimensional space. The centroid vector of a set of n relevant document vectors (r_1, r_2, .. r_n) is defined as:

$$Centroid = \frac{1}{n}\sum_{i=1}^{n} r_i. \tag{3.1}$$

Next, the average cosine between each relevant document vector and its corresponding centroid is calculated. The results are shown in Table 3.1. The *Relv. Doc.* field shows the number of relevant documents provided by OHSUMED for any given query. The *Avg. Cos* $\pm$ *Std. Dev.* field shows the average cosine between each relevant document vector and its corresponding centroid. A high cosine value indicates that angles between relevant document vectors and the centroid are relatively small, and suggests that relevant documents are tightly-clustered around the centroid.

In order to act as a proxy, a newly added vector should be close to one of the centroid vectors. It is also necessary that the semantic structures (UMLS and MeSH) organize knowledge at a granularity similar to the relevant clusters targeted by the query. Granularity is determined by how broad or narrow the semantic structures are constructed.

Table 3.1: Average cosine between each relevant document and the centroid for 106 queries.

	Relv. Docs.	Avg. Cos ± Std. Dev.		Relv. Docs.	Avg. Cos ± Std. Dev.		Relv. Docs.	Avg. Cos ± Std. Dev.
1	24	0.871 ± 0.141	37	36	0.678 ± 0.138	73	4	0.904 ± 0.039
2	11	0.720 ± 0.113	38	7	0.848 ± 0.101	74	21	0.805 ± 0.067
3	118	0.757 ± 0.113	39	2	0.938 ± 0.030	75	30	0.719 ± 0.081
4	4	0.745 ± 0.054	40	9	0.735 ± 0.079	76	7	0.838 ± 0.031
5	18	0.792 ± 0.112	41	18	0.711 ± 0.136	77	14	0.749 ± 0.093
6	30	0.930 ± 0.038	42	14	0.918 ± 0.052	78	9	0.829 ± 0.075
7	4	0.919 ± 0.019	43	49	0.782 ± 0.068	79	37	0.605 ± 0.112
8	0		44	7	0.708 ± 0.128	80	15	0.855 ± 0.064
9	5	0.739 ± 0.237	45	4	0.844 ± 0.071	81	5	0.801 ± 0.092
10	5	0.775 ± 0.156	46	35	0.839 ± 0.079	82	45	0.819 ± 0.115
11	24	0.737 ± 0.073	47	29	0.817 ± 0.091	83	49	0.813 ± 0.095
12	3	0.932 ± 0.059	48	9	0.632 ± 0.152	84	26	0.815 ± 0.089
13	12	0.772 ± 0.097	49	0		85	1	1.000 ± 0.000
14	10	0.787 ± 0.048	50	31	0.796 ± 0.100	86	0	
15	6	0.811 ± 0.076	51	4	0.667 ± 0.095	87	6	0.908 ± 0.036
16	47	0.687 ± 0.104	52	12	0.743 ± 0.155	88	50	0.874 ± 0.087
17	23	0.815 ± 0.121	53	63	0.713 ± 0.139	89	8	0.864 ± 0.068
18	15	0.841 ± 0.074	54	94	0.826 ± 0.098	90	6	0.777 ± 0.096
19	2	0.878 ± 0.076	55	24	0.719 ± 0.114	91	10	0.735 ± 0.095
20	1	1.000 ± 0.000	56	3	0.893 ± 0.039	92	6	0.924 ± 0.039
21	6	0.708 ± 0.180	57	29	0.656 ± 0.140	93	0	
22	76	0.899 ± 0.058	58	83	0.883 ± 0.089	94	26	0.882 ± 0.062
23	5	0.721 ± 0.056	59	11	0.733 ± 0.135	95	13	0.764 ± 0.115
24	2	0.727 ± 0.238	60	4	0.910 ± 0.028	96	24	0.726 ± 0.110
25	5	0.756 ± 0.071	61	4	0.923 ± 0.020	97	15	0.849 ± 0.050
26	19	0.719 ± 0.053	62	79	0.629 ± 0.150	98	6	0.803 ± 0.045
27	46	0.646 ± 0.092	63	47	0.709 ± 0.145	99	34	0.834 ± 0.071
28	0		64	52	0.671 ± 0.121	100	2	0.972 ± 0.008
29	28	0.808 ± 0.066	65	32	0.732 ± 0.106	101	15	0.927 ± 0.053
30	11	0.821 ± 0.082	66	6	0.662 ± 0.172	102	14	0.795 ± 0.126
31	13	0.848 ± 0.071	67	96	0.783 ± 0.098	103	15	0.770 ± 0.056
32	18	0.727 ± 0.119	68	7	0.779 ± 0.144	104	3	0.890 ± 0.039
33	26	0.892 ± 0.080	69	39	0.762 ± 0.100	105	12	0.760 ± 0.092
34	14	0.824 ± 0.064	70	8	0.835 ± 0.052	106	49	0.782 ± 0.069
35	63	0.751 ± 0.096	71	6	0.813 ± 0.069			
36	1	1.000 ± 0.000	72	27	0.813 ± 0.155			

3.1 KELSI with UMLS Semantic Network

3.1.1 Query Replacement

Concept vectors ($\vec{cv}$) built from the UMLS semantic network can be added as columns to the original term-by-document matrix. To see if they can serve as proxies, the concept vectors are matched against the centroid vectors (corresponding to the sets of relevant documents).

For each query, the concept vector that is the best match to its centroid vector is selected. If the concept vectors can serve as proxies, the best-matched concept vector should be close to the centroid and should return good results when used for query replacement.

The LSI++ software environment [LB97] is used to match queries and generate ranked returns. Specifically, a query is initially transformed into a vector in the reduced-dimensional space, and then matched against all document vectors to generate a ranked list of documents. This ranking is determined by the cosine between query and the document vectors. After obtaining a ranked list of documents, a series of post-processing scripts are used to compare the ranked list against the relevant documents for that particular query, calculate precision-recall values, tabulate and graph results.

Search results can be represented in either graphical or tabular formats. The graphical format is comprised of interpolated precision-recall plots [BYRN99].

Specifically, the interpolated precision values at eleven standard recall points (0.0, 0.1, 0.2, ..., 1.0) are plotted. These interpolated precision values are based on the *pseudo-precision* ($\tilde{P}$) [BB99]:

$$\tilde{P} = max\ P_i,\ where\ x \leq \frac{r_i}{r_n},\ and\ i = 1, 2, ..., n,$$

where r_i denotes the number of relevant documents up to and including position i in the ordered returned list of documents, and P_i is the precision at the i-th document. P_i is defined to be the proportion of documents up to and including position i that are relevant to the given query. The 11-point average precision values (P) are calculated by taking an average of $\tilde{P}$ at the standard recall points with $n = 11$:

$$P = \frac{1}{n} \sum_{i=0}^{n-1} \tilde{P} \left(\frac{i}{n-1} \right).$$

P values are considered as concise representations of their corresponding interpolated precision-recall graphs. The tabular format is simply a listing of 11-point average precision values obtained from KELSI versus that obtained from the original LSI method.

As a primary goal of this study is to use external semantic structures for enhancing query performance, overall performance gain is measured for a baseline of precision-recall data obtained from the original LSI method. In all graphs presented, the results from the original LSI method are represented by dashed lines and results from KELSI are represented by solid lines. For any query, if the P value from KELSI is larger than that from the baseline, its corresponding graph is shaded. For visible improvements, only those queries with $P_{KELSI} > .01$ are shaded.

The precision-recall graphs for this round of searching are presented in Figure B.1 in Appendix B. There is not a single shaded graph in that figure – indicating that the method of query replacement is very inefficient. The reason for this can be attributed to the fact that the UMLS Semantic Network has only 134 nodes. They often represent broad categories. For example, both query 5 (`effectiveness of etidronate in treating hypercalcemia of malignancy`) and query 88 (`lung cancer, radiation therapy`) are mapped to the concept vector *Fully Formed Anatomical Structure.*

3.1.2 Adding Concept Vectors

The ineffectiveness of concept vectors to act as proxies does not preclude them from redirecting the query vector toward the cluster of relevant documents. In this case, they are used as *guides* by incoming queries. The investigation precedes as follows: first, find the best matched concept vectors ($\vec{cv}$), and modify the query vector ($\vec{q}$) as:

$$\vec{q_{add}} = \vec{q} + \vec{cv}. \tag{3.2}$$

The effect of the modification is to redirect the query vector in the direction of the concept vector. Next, the modified query is used to carry out the search. Results are displayed in Figure B.2 in Appendix B. Here, there are only 134 concept vectors and they have on average 1,804 words. In contrast, the queries have on average only 7 words. So, the concept vector tends to dominate the direction of $\vec{q_{add}}$ and it effectively amounts to query replacement in most cases. Recall that query replacement has not performed well.

3.1.3 Adding Projections

Having seen that query modification by concept vector addition does not perform well, reducing the effect of query modification might be desirable. Instead of adding the entire concept vector, only the projection of the query vector onto the concept vector ($proj_{\vec{cv}}$) could be added to the original query. In other words,

$$\vec{q_{proj}} = \vec{q} + proj_{\vec{cv}}\, \vec{q} = \vec{q} + \frac{\vec{cv}^T \vec{q}}{\vec{cv}^T \vec{cv}}\, \vec{q}. \tag{3.3}$$

This still redirects the query vector in the direction of concept vector but the query modification is much more modest. Again, the modified queries are used to search the collection.

The results of this round of searching is presented in Figure 3.1. There is noticeable improvement for some queries (illustrated by shaded precision-recall graphs). Performance comparisons for the three approaches considered (based on P values) will be covered later in this chapter.

3.2 KELSI with MeSH Headings

As discussed in Section 2.5, adding MeSH headings introduces an additional 13,853 row vectors into the original term-by-document matrix. Again, the centroids (see Equation 3.1) are used to determine how well the MeSH headings can serve as proxies for the original queries. The centroids are used to extract the best-matched MeSH vectors, which are then used as query replacements. Search results are presented in Figure 3.2. The centroids returned better results (in terms of P values) in 63 out of 106 queries. Visual inspection of the figure reveals that significant improvements in precision are achieved for many queries (detailed analysis is deferred until later). This set of results demonstrates that MeSH vectors can be highly effective when used as search proxies.

The centroid approach involves working backward from the set of relevant documents to establish the fact that MeSH vectors can serve as effective proxies. So, the question becomes how can the best MeSH heading for each query be identified without help from the centroid vectors (corresponding to the *known*

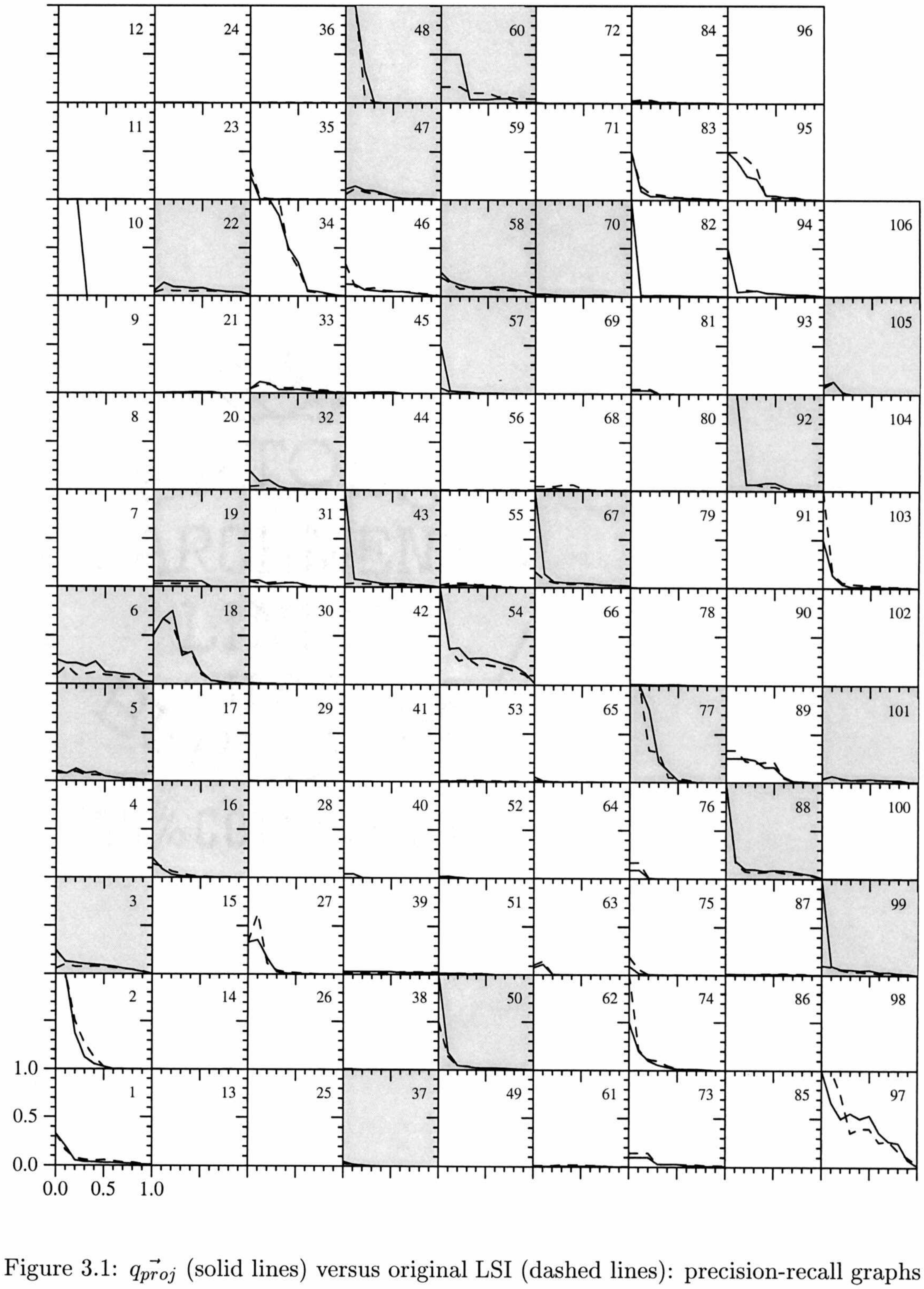

Figure 3.1: $\vec{q_{proj}}$ (solid lines) versus original LSI (dashed lines): precision-recall graphs

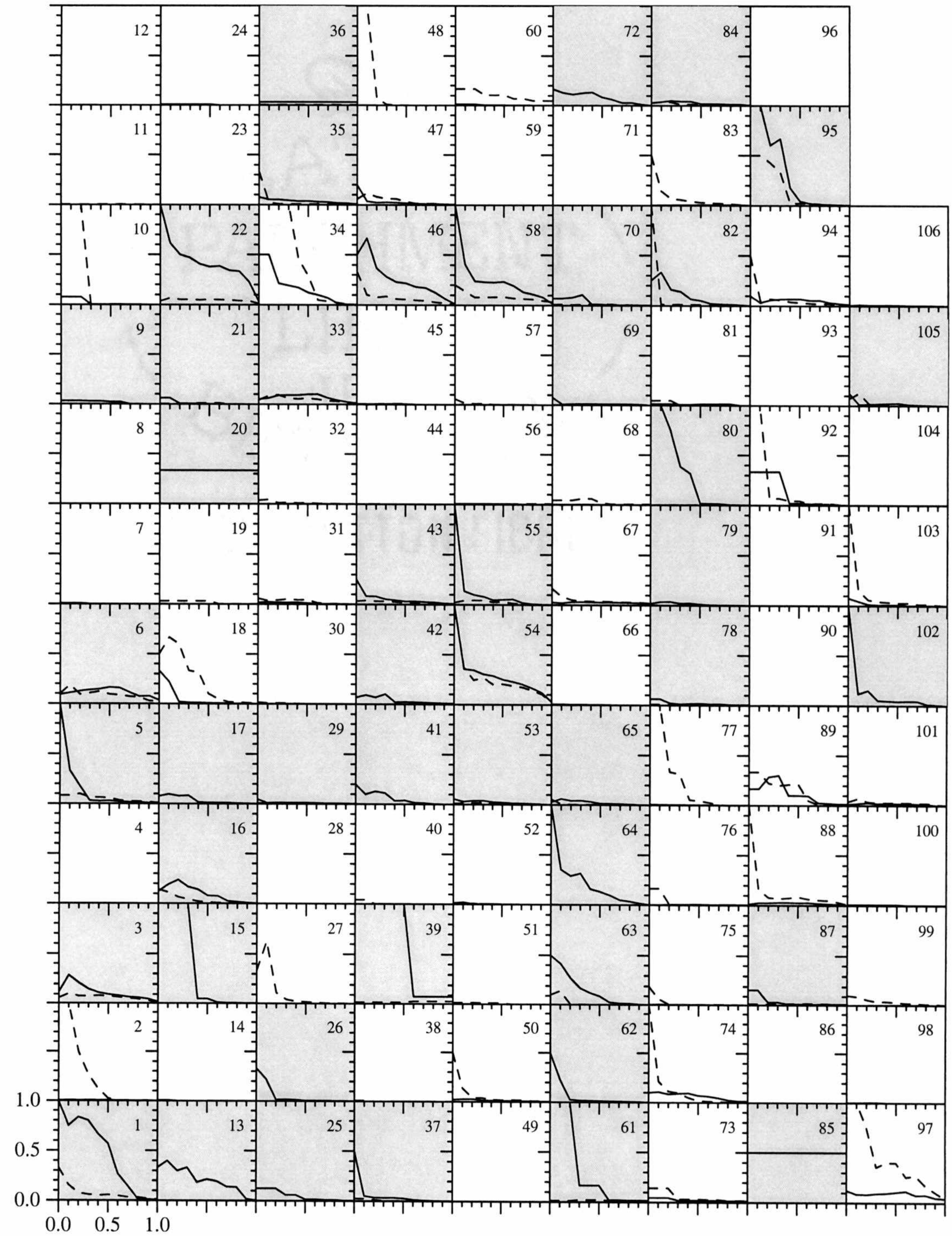

Figure 3.2: MeSH vectors nearest to centroids (solid lines) versus LSI (dashed lines): precision-recall graphs

relevant documents).

Three approaches are attempted:

1. Direct query approach: conduct direct query match against the 13,853 newly added row vectors in the MeSH enhanced search space, pick the top ranked MeSH headings as proxies.

2. Two-step approach: In Step 1, construct a small term-by-heading matrix S using the MeSH definition files presented in Chapter 2. The best-matched MeSH headings are then selected by matching queries against a low-rank approximation (S_k) to matrix S using LSI. In Step 2, use those heading to search the larger MeSH enhanced term-by-document matrix.

3. Regular expression approach: utilize the thesaurus property of the MeSH headings to extract MeSH headings directly from the query string using regular expression match.

3.2.1 Direct Query Match

In the first approach, the search space has already been constructed and queries are fixed. There is very little flexibility. The precision-recall graphs (Figure C.1 in Appendix C) show that finding the best MeSH vector by direct query matching is not very effective. Again, specific performance comparisons are deferred to the end of this chapter.

3.2.2 Two-step Approach

Results from two-step approach are shown in Figure C.2 in Appendix C. Those results indicate that this approach yields very little improvement. The description fields of MeSH headings are used to construct the smaller term-by-heading matrix. Those fields are usually short (33 words on average). With limited scope for term co-occurrence, and concept specificity. LSI does not perform well in this particular case [Kow97].

3.2.3 Regular Expression Match

The third approach utilizes the thesaurus property of the MeSH Headings. The *ENTRY* fields (see Table A.3 in Appendix A) in the MeSH description maps alternate spellings and synonyms into a MeSH heading. Regular expression matching can be used to extract these headings within query strings.

It is possible that more than one MeSH heading can be extracted from a single query. In those cases, the MeSH tree structure is used to select the one with the smallest granularity, i.e., furthest from the root. For this purpose, each heading is associated with a number indicating how many levels removed it is from the top. The heading with the largest number is selected. In case of ties, the first one gets selected. In addition, headings from the top two levels are rejected as they are typically very broad (or abstract). Abstract concepts, as shown earlier in UMLS semantic nodes, often perform poorly as proxies. Finally, there is a stop-list of 25 very broad MeSH headings that include *disease, syndrome*, and *pain* etc. Those are general concepts that are located outside of the top two levels of MeSH tree.

To illustrate this approach using an example, consider query 37:

`Fibromyalgia/fibrositis, diagnosis and treatment.`

Regular expression match extracted the following MeSH headings (Table 3.2):

Table 3.2: MeSH regular expression match for query 37

Query Word(s) → MeSH Heading	MeSH Tree ID	Distance to Root
diagnosis → diagnosis	E01	1
fibromyalgia → fibromyalgia	C10	4
fibrositis → fibromyalgia	C10	4
treatment → therapeutics	E02	1

The MeSH heading the furthest from the root is *fibromyalgia*. It appears in the MeSH tree *C10* (see Table 3.3).

Following the methods mentioned above, MeSH headings are found for 89 out of the 106 possible queries. Fourteen of them match the best MeSH headings selected by the centroids. All 89 are used as proxies for their respective queries and the search results are displayed in Figure 3.3. Queries for which the MeSH enhanced method did better (i.e., higher P values were obtained) are lightly shaded. Queries for which the MeSH headings based on regular expression matching performed the same as the version based on centroid matching are denoted by a darker shade.

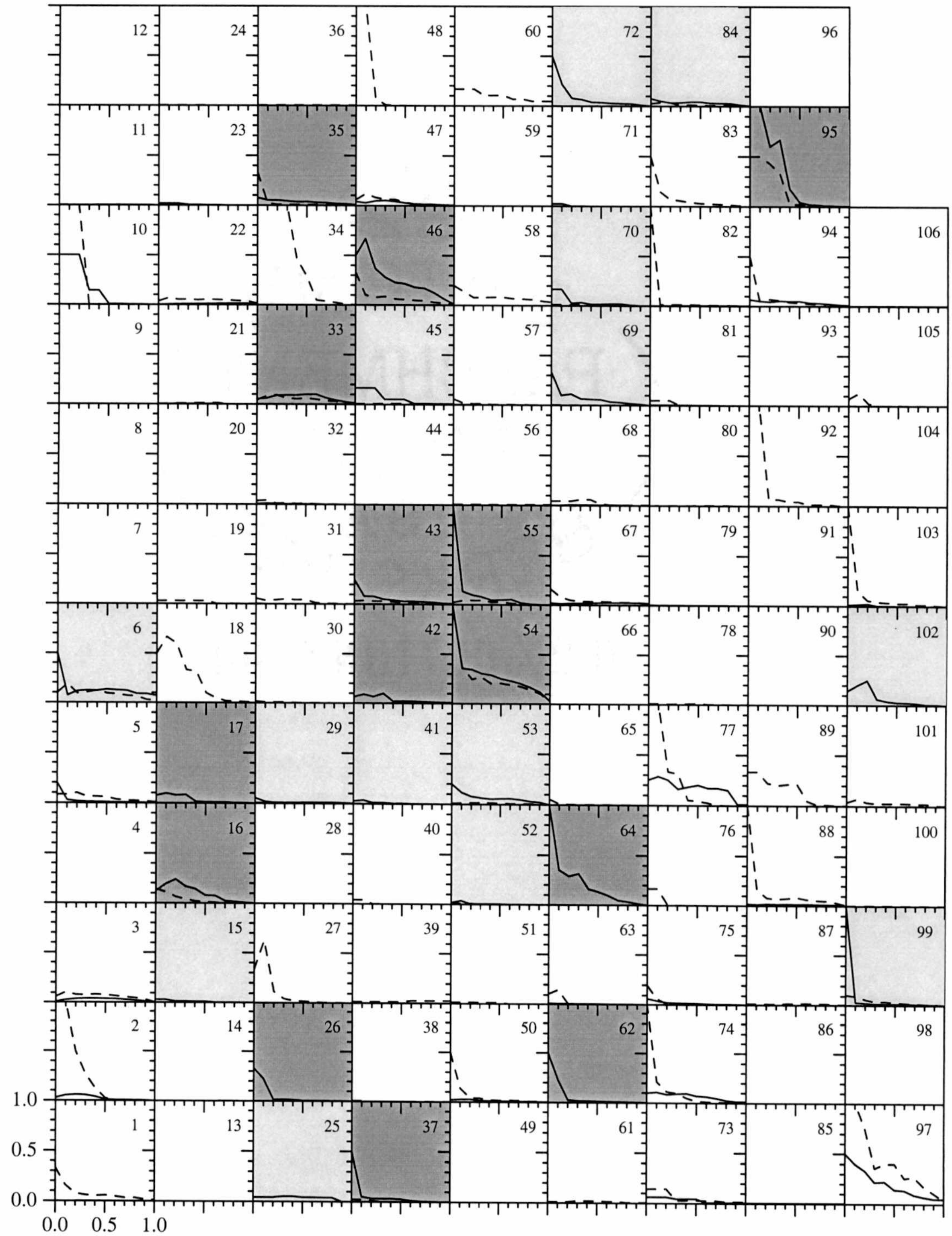

Figure 3.3: Precision-recall graphs: regular expression matched MeSH vectors (solid lines) versus LSI (dashed lines)

Table 3.3: Location of *Fibromyalgia* within the MeSH tree *C10.*

Nervous System Diseases [C10]
.. Neuromuscular Diseases [C10.668]
.... Muscular Diseases [C10.668.491]
...... Muscular Disorders, Atrophic [C10.668.491.175] +
.... Eosinophilia-Myalgia Syndrome [C10.668.491.387]
.... **Fibromyalgia [C10.668.491.425]**
.... Mitochondrial Myopathies [C10.668.491.500] +
.... Myopathies, Structural, Congenital [C10.668.491.550] +
.... Myositis [C10.668.491.562] +
.... Myotonic Disorders [C10.668.491.606] +
.... Paralyses, Familial Periodic [C10.668.491.650]

3.3 Performance Comparison

The performance of KELSI with three approaches for exploiting UMLS concept vectors and three approaches for including MeSH headings has been studied. All approachs are evaluated on a query-by-query basis, the 11-point average precision values from the new methods against those from the original LSI. Such a crude comparison would give a first-order indication of how effective the new methods are.

As illustrated in Table 3.4, the vector projection approach returns the best results for KELSI with UMLS concept vectors. At the same time, the regular expression match approach returns the best results for KELSI with MeSH headings. Subsequent analysis will focus on those two approaches.

Table 3.4: Performance comparison for UMLS concept vectors and MeSH headings

	Versions of KELSI Queries	Queries with improvement over LSI (Out of 106 Queries)
UMLS	Query Replacement	0
	Add Concept Vector	9
	Add Vector Projection	44
MeSH	Direct Match	19
	Two-Step Match	3
	Regular Expression Match	37

Chapter 4

Incremental Performance Gain

As discussed in the previous chapter, out of a total of 106 queries, the LSI enhancement based on UMLS concept vectors did better (in terms of average precision) than the original LSI method for 44 queries. The LSI enhancement exploiting MeSH headings did better than the original LSI method for 35 queries. Those results cloud the prospect of using the new methods as replacements for LSI. However, there remains the possibility that they can be used as complements to the original LSI approach. Subsequent analysis is focused on the subset of queries where the new methods performed better. That is, the identification of queries which show significant improvement (in precision) for the new methods but limited or poor retrieval by LSI.

For notational convenience, LSI_{UMLS} is used to refer to results obtained from LSI enhanced by UMLS concept vectors and LSI_{MeSH} is used for LSI enhanced by MeSH vectors.

4.1 Magnitude of Improvement

Table 4.1 shows the magnitude of improvement for queries that have produced larger P values under the enhanced methods than under the original LSI method. The magnitude of improvement is calculated as

$$Improvement\ Factor = \frac{P_{[UMLS|MeSH]} - P_{LSI}}{P_{LSI}},$$

where P_{UMLS} or P_{MeSH} is the 11-point interpolated average precision (see Chapter 3) obtained by LSI enhanced with UMLS or MeSH. Notice that at the lower end of the improvement scale (.1 or .2), LSI_{UMLS} and LSI_{MeSH} perform better than LSI on a comparable number of queries. However, at the upper end, LSI_{MeSH} does noticeably better than LSI_{UMLS}. For example, LSI_{MeSH} did twice as well as LSI on 27 queries, versus 5 for LSI_{UMLS}.

4.2 Incremental Performance Gain

To be an effective complement to LSI, any enhanced method should return good results for queries where LSI does poorly, i.e., there should be some incremental performance gain. One way to evaluate this performance gain is from the user's

Table 4.1: Magnitude of improvement for new methods which performed better.

Improvement Factor	LSI_{UMLS} Better (No. of Queries)	LSI_{MeSH} Better (No. of Queries)
.1	36	34
.2	27	32
.3	20	31
.4	16	29
.5	12	29
1	5	27
10	0	13
100	0	8

perspective. Here, the user specifies a threshold of relevance based on an 11-point average precision (P) value, and the original LSI method is then evaluated at each threshold level. For all 106 queries, the queries that meet the threshold are accepted and all others are collected into a rejected pool. Next, the user examines results of LSI_{UMLS} and LSI_{MeSH} for queries in the rejected pool. Queries that meet the threshold are accepted and signify an incremental performance gain.

The thresholds selected for this analysis are: .30, .20, .10, .05, and .01. Tables 4.2 and 4.3 show two examples of a ranked list of relevant documents from LSI_{MeSH} that correspond to the low and high ends of the threshold scale: query 15 with $P = .0125$ and query 95 with $P = .3285$. *Document ID* is the unique OHSUMED ID of a relevant document. *Rank* denotes the position of those relevant documents in the ranked list returned by query matching.

Table 4.4 shows the incremental performance gain obtained for the LSI_{UMLS}

Table 4.2: Ranked return set for query 15: $P = .0125$

Document ID	Rank
86570	30
300573	127
207730	241
321704	1024
140596	1408
238356	3803

Table 4.3: Ranked return set for query 95: $P = .3285$

Document ID	Rank
3343	1
62473	2
81232	5
12873	6
119850	10
99052	34
54393	157
278133	351
36726	697
265924	839
181481	960
162561	1736
266317	2124

Table 4.4: Incremental performance gain from LSI_{UMLS} and LSI_{MeSH} – number of queries accepted for each threshold.

User Specified Threshold (P)	LSI Accept	Out of LSI Reject Pool	
		LSI_{UMLS} Accept	LSI_{MeSH} Accept
.30	2	1	2
.20	8	1	3
.10	17	6	5
.05	30	4	13
.01	53	1	15

and LSI_{MeSH} methods. The number of queries accepted by the original LSI method is shown first, followed by the number of queries accepted out of the rejected pool by LSI_{UMLS} and LSI_{MeSH}. The number of acceptable queries for the original LSI method is cumulative, whereas the number of queries accepted by the enhanced methods are not – the rejected pool shrinks as the threshold is progressively lowered.

Table 4.5 shows the detailed breakdown (by threshold) of queries accepted by LSI_{UMLS} out of the rejected pool. Similarly, Table 4.6 shows the detailed breakdown (by threshold) of queries accepted by LSI_{MeSH} out of the rejected pool. Comparing those two tables, there are striking differences in both magnitude of improvement and in number of queries accepted. At its best, LSI_{UMLS} delivered improvement over original LSI by a factor of 3.92 – a value that was exceeded 26 times in LSI_{MeSH}. In addition, summing up across all thresholds, LSI_{UMLS} picked up 13 additional queries, whereas LSI_{MeSH} picked up 38.

Table 4.5: Incremental performance gain for LSI_{UMLS}.

Threshold	Query Number	P_{LSI}	P_{UMLS}	Factor
0.3	54	0.2684	0.3081	0.15
0.2	48	0.1894	0.2134	0.13
0.1	**43**	0.0261	**0.1285**	**3.92**
0.1	50	0.0754	0.1247	0.65
0.1	58	0.0876	0.1054	0.20
0.1	60	0.0995	0.1627	0.63
0.1	67	0.0590	0.1378	1.34
0.1	**99**	0.0424	**0.1357**	**2.20**
0.05	**43**	0.0261	**0.1285**	**3.92**
0.05	47	0.0485	0.0610	0.26
0.05	**57**	0.0114	**0.0539**	**3.71**
0.05	**99**	0.0424	**0.1357**	**2.20**
0.01	37	0.0065	0.0100	0.53

These results indicate that LSI_{MeSH} is far superior than LSI_{UMLS} when it comes to delivering incremental performance gain over original LSI. Here are a few examples that demonstrate the improvement in precision achieved by LSI_{MeSH}.

`Query 64: prevention, risk factors, pathophysiology of hypothermia`

LSI_{MeSH} replaced this query with the MeSH heading *hypothermia*. The resulting $P = .2290$, versus $P = .0007$ for the original LSI method – a 320-fold improvement. (see Table 4.6). In this case, the query reflects a single concept, hypothermia, and it is mapped to the corresponding MeSH heading.

`Query 55: course of anticoagulation with coumadin`

LSI_{MeSH} replace this query by the MeSH heading *warfarin* and improved the

Table 4.6: Incremental performance gain for LSI_{MeSH}.

Threshold	Query Number	P_{LSI}	P_{MeSH}	Factor
0.3	54	0.2684	0.3024	0.13
0.3	95	0.1703	0.3285	0.93
0.2	46	0.0821	0.2724	2.32
0.2	**64**	0.0007	**0.2290**	**320.80**
0.2	95	0.1703	0.3285	0.93
0.1	16	0.0388	0.1234	2.18
0.1	46	0.0821	0.2724	2.32
0.1	55	0.0248	0.1382	4.58
0.1	**64**	0.0007	**0.2290**	**320.80**
0.1	99	0.0424	0.1071	1.52
0.05	16	0.0388	0.1234	2.18
0.05	26	0.0011	0.0581	50.58
0.05	37	0.0065	0.0659	9.09
0.05	43	0.0261	0.0603	1.31
0.05	45	0.0065	0.0643	8.92
0.05	53	0.0120	0.0760	5.33
0.05	55	0.0248	0.1382	4.58
0.05	**62**	0.0005	**0.0921**	**188.63**
0.05	**64**	0.0007	**0.2290**	**320.80**
0.05	69	0.0019	0.0766	39.43
0.05	72	0.0017	0.0950	54.32
0.05	99	0.0424	0.1071	1.52
0.05	**102**	0.0007	**0.0859**	**124.05**
0.01	**15**	0.0001	**0.0125**	**120.43**
0.01	**17**	0.0003	**0.0445**	**163.34**
0.01	**25**	0.0001	**0.0398**	**355.60**
0.01	26	0.0011	0.0581	50.58
0.01	29	0.0044	0.0161	2.68
0.01	37	0.0065	0.0659	9.09
0.01	41	0.0010	0.0109	9.61
0.01	**42**	0.0002	**0.0479**	**282.46**
0.01	45	0.0065	0.0643	8.92
0.01	52	0.0036	0.0100	1.80
0.01	**62**	0.0005	**0.0921**	**188.63**
0.01	**64**	0.0007	**0.2290**	**320.80**
0.01	69	0.0019	0.0766	39.43
0.01	72	0.0017	0.0950	54.32
0.01	**102**	0.0007	**0.0859**	**124.05**

average precision by a factor of 4.58 (P = .1382, versus P = .0248 over the original LSI method). In this case, the thesaurus property (see Chapter 3 Section 3.2.3) of MeSH is utilized to map *coumadin* directly to *warfarin.*

```
Query 71: cystic fibrosis and renal failure, effect of long term
          repeated use of aminoglycosides
```

Regular expression matching produced the following replacements: (values shown are distances to the tree root, see Section 3.2.3)

```
aminoglycosides  --> aminoglycosides  4
cystic fibrosis  --> cystic fibrosis  3
renal failure    --> kidney failure  4
```

Here the situation becomes murkier. The query involves several interrelated concepts, but only one is selected for query replacement (by design). LSI_{MeSH} uses the MeSH heading farthest from the root, so the choice is between *aminoglycosides* and *kidney failure. Aminoglycosides* is arbitrarily chosen. This results in P = 0.0054 versus P = 0.0004 for the original LSI method, hence, a 12-fold improvement. Still this query cannot make even the lowest threshold.

Chapter 5

Summary and Conclusions

In summary, KELSI search indices are built by augmenting a term-by-document matrix with vectors constructed from UMLS semantic network and MeSH headings. During query matching, original queries are either modified, as in the case of UMLS concept vector, or replaced as in the case of MeSH headings. Results based on precision-recall graphs and P values show that LSI_{MeSH} is superior to LSI_{UMLS} with respect to incremental performance gains over the original LSI model. For P=.30, LSI_{MeSH} delivered 100% (2/2) incremental improvement over the original LSI versus 50% (1/2) for LSI_{UMLS}. For P=.01, LSI_{MeSH} delivered 28% (15/53) incremental improvement over the original LSI. versus .19% (1/53) for LSI_{UMLS} (Table 4.4). The effectiveness of LSI_{MeSH} can be attributed to three factors:

1. Concepts within documents from the OHSUMED collection are unambigu-

ously identified by MeSH headings. This simplifies the process of incorporating semantic structures into an existing LSI model (Section 2.5).

2. MeSH headings organize or categorize information in granularities similar to that targeted by the queries. This is certainly necessary for query replacement to be effective (Figure 3.2).

3. The thesaurus functionality of MeSH headings is instrumental in mapping query text into MeSH headings through regular expression matching. The tree structure of MeSH headings is especially helpful for narrowing the scope of a query, i.e., traversing the hierarchy to find the most specific heading (Section 3.2.3).

In contrast, the UMLS semantic tree categorizes knowledge at more abstract levels. In addition, there is no explicit information (metadata) within OHSUMED documents that identify UMLS concepts, and to identify them externally involves cumbersome mappings (Section 2.4).

It is possible that the incremental performance gains from LSI_{MeSH} can be delivered for a relatively small computation cost during query matching. This can be done by matching each MeSH heading against the search index a priori and caching the results. Later, when an user's query is entered, cached results could be returned as soon as a query replacement is found.

As shown in Figure 3.2, LSI_{MeSH} is capable of delivering much better incremental performance gains over the original LSI model. Finding the best possible MeSH headings for query modification is the challenge. The regular expression approach used in this study is straightforward and effective but it does leave plenty of room for improvement.

For this study, it is very fortunate that the MeSH headings happen to organize information at granularities similar to that targeted by the queries. It is even more fortunate that OHSUMED documents are directly linked to the MeSH headings. KELSI's application will be very limited if its effectiveness hinges on such good fortunes. The two main factors that can potentially limit KELSI's general application are: the requirement of an external semantic structure that organizes information at granularities similar to that targeted by user queries, and direct linkage (perhaps by metadata) between the documents and that semantic structure.

The emerging web standard – Extensible Markup Language (XML) [BPSMM00] – holds immense promise for clearing these hurdles. XML documents can explicitly identify themselves to external semantic structures through semantic markup tags. Those tags are defined in external semantic structures such as the Document Type Definition (DTD) [BPSMM00] or some other framework. Should a DTD be designed with user queries in mind, it may be possible to use them as external semantic structures in KELSI to improve search performance.

In conclusion, this study has shown that external semantic structures can be incorporated into the original LSI model to produce enhanced search indices. Query modification and query replacement methods can then be applied during query matching to exploit the enhanced search indices. In the case of LSI_{MeSH}, noticeable incremental performance gains over the original LSI were achieved.

Bibliography

Bibliography

[BB99] MW Berry and M Browne. *Understanding Search Engines: Mathematical Modeling and Text Retrieval.* SIAM Book Series: Software, Environments, and Tools, first edition, 1999.

[BPSMM00] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, and Eve Maler. Extensible Markup Language (XML) 1.0 (Second Edition). *W3C Recommendation*, `http://www.w3.org/TR/REC-xml`, 2000.

[BYRN99] Richardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval.* Addison Wesley, first edition, 1999.

[DDF+90] Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman. Indexing by Latent Semantic Analysis. *Journal of the American Society for Information Science*, 41(2):391–407, 1990.

[GL96] Gene H. Golub and Charles F. Van Loan. *Matrix Computations.* Johns Hopkins University Press, Baltimore, MD, third edition, 1996.

[HBLH94] W Hersh, C Buckley, T Leone, and D Hickam. OHSUMED: An interactive retrieval evaluation and new large test collection for research. *Proceedings of the 17th Annual ACM SIGIR Conference*, pages 192–201, 1994.

[HTBM00] S Howard, H Tang, M Berry, and D. Martin. *General Text Parser GTP Version 2.0.* Department of Computer Sciences, The University of Tennessee, `http://www.cs.utk.edu/~lsi/`, 2000.

[Kow97] G Kowalski. *Information Retrieval Systems: Theory and Implementation.* Kluwer Academic Publishers, Boston, 1997.

[LB97] Todd A. Letsche and Michael W. Berry. Large-Scale Information Retrieval with Latent Semantic Indexing. *Information Sciences - Applications*, 100:105 – 137, 1997.

[NBB+00] S Nelson, S Bear, M Belony, D Johnston, J Pash, T Powell, A Savage, J Schulman, N Sorden, and L Tang. *Medical Subject Headings.* National Library of Medicine, `http://www.nlm.nih.gov/ mesh/meshhome.html`, 2000.

[SHB97] C Selden, R Humphreys, and L Betsy. *Unified Medical Language System (UMLS).* U.S. Department of Health and Human Services, Public Health Service, National Institutes of Health, `http://umlsinfo.nlm.nih.gov/`, 1997.

Appendices

Appendix A

Sample Entries from OHSUMED and MeSH

Sample entries from the OHSUMED collection and MeSH headings are included in this section.

1. Table A.1 shows a sample entry from the OHSUMED collection.

2. Table A.2 is a list of data fields for the OHSUMED collection.

3. Table A.3 is a sample entry from the MeSH heading description file.

4. Table A.4 is a list of top-level MeSH trees.

Table A.1: A sample entry from the OHSUMED collection.

```
.I
125536
.U
89315773
.S
Proc Natl Acad Sci U S A 8910; 86(14):5242-6
.M
Amino Acid Sequence; Animal; Argipressin/GE; Base Sequenc e; Cloning,
Molecular/*; Comparative Study; DNA/*GE; DNA Polymerases; Fishes/*GE;
Gene Amplification; Genes, Structural; Human; Molecular Sequence Data;
Oxytocin/*AA/GE; Protein Precursors/*GE; Sequence Homology,
Nucleic Acid; S upport, Non-U.S. Gov't; Vasotocin/*GE.
.T
Vasotocin and isotocin precursors from the white sucker, Catostomus
commersoni: cloning and sequence analysis of the cDNAs.
.P
JOURNAL ARTICLE.
.W
The nucleotide sequences of cloned cDNAs encoding the pre cursors
for vasotocin and isotocin have been elucidated by analyzing a lambda
gt11 library constructed from poly(A)+ RNA from the hypothalamic region
of the teleost fish Catostomus commersoni. Screening of the library was
carried out with synthetic oligonucleotide probes deduced from the amino
acid sequences of the nonapeptides vasotocin and isotocin. The cDNA
nucleotide sequences predict isotocin and vasotocin prohormone precursors
each consisting of a signal peptide, a hormone moiety, and a
neurophysin-like molecule. However, in comparison to their mammalian
counterparts, both fish neurophysins are extended at their C termini
by an approximately 30 amino acid sequence with a leucine-rich core
segment. These extensions show striking similarities with the glycopeptide
moiety (the so-called copeptin) present in mammalian vasopressin precursors,
except that they lack the consensus sequence for N-glyco sylation. These
data suggest that mammalian copeptin is derived from the C terminus of
an ancestral neurophysin.
.A
Heierhorst J; Morley SD; Figueroa J; Krentler C; Lederis
K; Richter D.
```

Table A.2: Data fields of an OHSUMED entry.

Entry Field	Field Definition
.I	Unique entry ID
.U	MEDLINE identifier
.M	Human-assigned MeSH terms
.T	Title
.P	Publication type
.W	Abstract
.A	Author
.S	Source

Table A.3: A sample entry from the MeSH heading description file.

```
*NEWRECORD
RECTYPE = D
MH = Viral Structural Proteins
AQ = AD AE AG AI AN BI BL CF CH CL CS CT DE DF DU EC GE
HI IM IP ME PD PH PK PO RE SD SE ST TO TU UL UR
ENTRY = Polypeptide VP1, Structural
ENTRY = Simian Virus 40 Virion Protein 1:
ENTRY = VP(1):T116:T123:ABB:NRW:NLM (1990):890208:abbcdef
ENTRY = VP(2):T116:T123:ABB:NRW:NLM (1990):890208:abbcdef
ENTRY = VP(3):T116:T123:ABB:NRW:NLM (1990):890208:abbcdef
ENTRY = VP(6):T116:T123:ABB:NRW:NLM (1990):890208:abbcdef
ENTRY = VP(7):T116:T123:ABB:NRW:NLM (1990):890208:abbcdef
ENTRY = Viral Structural Proteins
ENTRY = Proteins, Viral Structural
ENTRY = Structural Polypeptide VP1
ENTRY = Structural Proteins, Viral
ENTRY = VP1, Structural Polypeptide
MN = D12.776.964.970
MH_TH = NLM (1990)
ST = T116
ST = T123
RN = 0
AN = IM; coord with specific virus (IM); /drug effultrastruct permitted
PI = Viral Proteins (1973-1989)
MS = Viral proteins that do not regulate transcription. They are coded by viral
structural genes and include nucleocapsid core proteins (gag proteins), enzymes
(pol proteins), and membrane components (env proteins). Tran scription of viral
structural genes is regulated by viral regulatory proteins.
PM = 90
HN = 90
MED = *205
MED = 320
MR = 19950609
DA = 19890525
DC = 1
DX = 19900101
UI = D015678
```

Table A.4: A list of all top-level MeSH trees with the node *Chemicals and Drugs* expanded to the next level.

1. Anatomy [A]
2. Organisms [B]
3. Diseases [C]
4. Chemicals and Drugs [D]
.... Inorganic Chemicals [D01] +
.... Organic Chemicals [D02] +
.... Heterocyclic Compounds [D03] +
.... Polycyclic Hydrocarbons [D04] +
.... Environmental Pollutants, Noxae, and Pesticides [D05] +
.... Hormones, Hormone Substitutes, and Hormone Antagonists [D06] +
.... Reproductive Control Agents [D07] +
.... Enzymes, Coenzymes, and Enzyme Inhibitors [D08] +
.... Carbohydrates and Hypoglycemic Agents [D09] +
.... Lipids and Antilipemic Agents [D10] +
.... Growth Substances, Pigments, and Vitamins [D11] +
.... Amino Acids, Peptides, and Proteins [D12] +
.... Nucleic Acids, Nucleotides, and Nucleosides [D13] +
.... Neurotransmitters and Neurotransmitter Agents [D14] +
.... Central Nervous System Agents [D15] +
.... Peripheral Nervous System Agents [D16] +
.... Anti-Inflammatory Agents, Antirheumatic Agents, and Inflammation Mediators [D17] +
.... Cardiovascular Agents [D18] +
.... Hematologic, Gastrointestinal, and Renal Agents [D19] +
.... Anti-Infective Agents [D20] +
.... Anti-Allergic and Respiratory System Agents [D21] +
.... Antineoplastic and Immunosuppressive Agents [D22] +
.... Five more ...
5. Analytical, Diagnostic and Therapeutic Techniques and Equipment [E]
6. Psychiatry and Psychology [F]
7. Biological Sciences [G]
8. Physical Sciences [H]
9. Anthropology, Education, Sociology and Social Phenomena [I]
10. Technology and Food and Beverages [J]
11. Humanities [K]
12. Information Science [L]
13. Persons [M]
14. Health Care [N]
15. Geographic Locations [Z]

Appendix B

Results of Query Matching Using UMLS Concept Vectors

Results of query matching in Chapter 3, Section 3.1 are presented here. They include:

1. Precision-recall graphs for $\vec{cv}$ query replacement versus the original LSI method.

2. Table of $P_{\vec{cv}\ query\ replacement}$ versus P_{LSI}.

3. Precision-recall graphs for $q_{\vec{add}}$ versus the original LSI method.

4. Table of $P_{q_{\vec{add}}}$ versus P_{LSI}.

5. Table of $P_{q_{\vec{proj}}}$ versus P_{LSI}.

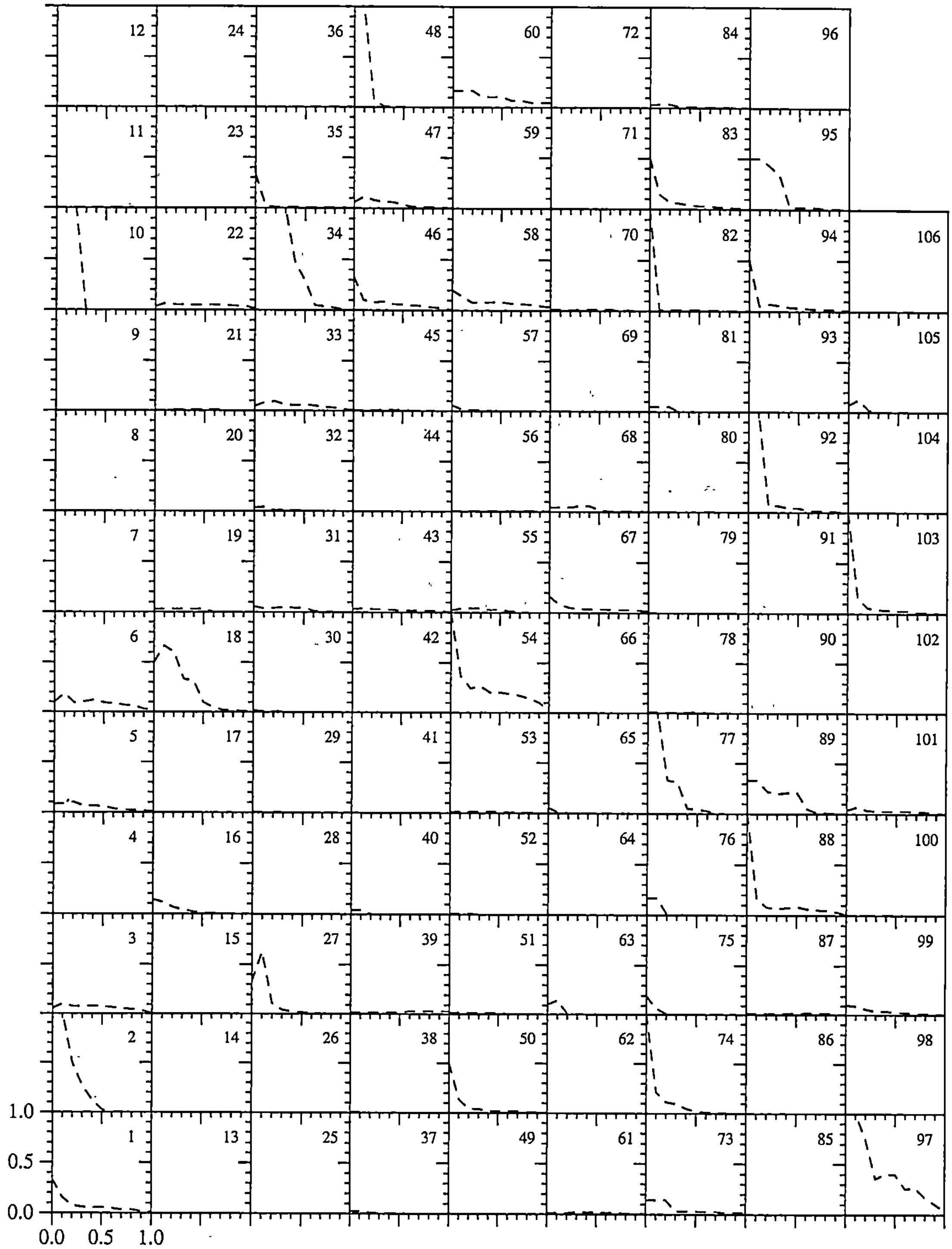

Figure B.1: Precision-recall graphs: $\vec{cv}$ query replacement versus original LSI.

Table B.1: $\vec{cv}$ query replacement versus original LSI: P values.

Query	P_{LSI}	P_{KELSI}	Query	P_{LSI}	P_{KELSI}	Query	P_{LSI}	P_{KELSI}
1	0.0863	0.0002	37	0.0065	0.0002	73	0.0567	0.0001
2	0.2657	0.0001	38	0.0033	0.0000	74	0.1388	0.0002
3	0.0714	0.0004	39	0.0303	0.0000	75	0.0280	0.0001
4	0.0001	0.0000	40	0.0081	0.0000	76	0.0318	0.0000
5	0.0557	0.0001	41	0.0010	0.0001	77	0.2584	0.0002
6	0.1007	0.0008	42	0.0002	0.0000	78	0.0073	0.0000
7	0.0010	0.0000	43	0.0261	0.0003	79	0.0007	0.0005
8	0.0000	0.0000	44	0.0007	0.0000	80	0.0025	0.0000
9	0.0002	0.0001	45	0.0065	0.0002	81	0.0143	0.0001
10	0.2734	0.0000	46	0.0821	0.0003	82	0.1001	0.0010
11	0.0046	0.0001	47	0.0485	0.0001	83	0.0847	0.0002
12	0.0000	0.0000	48	0.1894	0.0002	84	0.0170	0.0001
13	0.0003	0.0000	49	0.0000	0.0000	85	0.0001	0.0000
14	0.0012	0.0002	50	0.0754	0.0001	86	0.0000	0.0000
15	0.0001	0.0000	51	0.0089	0.0000	87	0.0167	0.0001
16	0.0388	0.0002	52	0.0036	0.0001	88	0.1557	0.0003
17	0.0003	0.0001	53	0.0120	0.0002	89	0.1456	0.0000
18	0.2584	0.0007	54	0.2684	0.0007	90	0.0006	0.0000
19	0.0176	0.0000	55	0.0248	0.0001	91	0.0003	0.0001
20	0.0006	0.0000	56	0.0075	0.0000	92	0.2077	0.0002
21	0.0085	0.0000	57	0.0114	0.0001	93	0.0000	0.0000
22	0.0511	0.0002	58	0.0876	0.0002	94	0.0796	0.0001
23	0.0001	0.0000	59	0.0003	0.0002	95	0.1703	0.0000
24	0.0015	0.0001	60	0.0995	0.0001	96	0.0009	0.0007
25	0.0001	0.0001	61	0.0161	0.0000	97	0.4390	0.0000
26	0.0011	0.0003	62	0.0005	0.0002	98	0.0003	0.0000
27	0.1348	0.0002	63	0.0295	0.0001	99	0.0424	0.0001
28	0.0000	0.0000	64	0.0007	0.0002	100	0.0036	0.0000
29	0.0044	0.0009	65	0.0085	0.0002	101	0.0318	0.0001
30	0.0049	0.0003	66	0.0005	0.0001	102	0.0007	0.0000
31	0.0302	0.0000	67	0.0590	0.0003	103	0.1261	0.0001
32	0.0135	0.0039	68	0.0319	0.0000	104	0.0010	0.0001
33	0.0613	0.0002	69	0.0019	0.0013	105	0.0262	0.0001
34	0.4432	0.0000	70	0.0108	0.0001	106	0.0038	0.0004
35	0.0383	0.0002	71	0.0004	0.0001			
36	0.0040	0.0000	72	0.0017	0.0001			

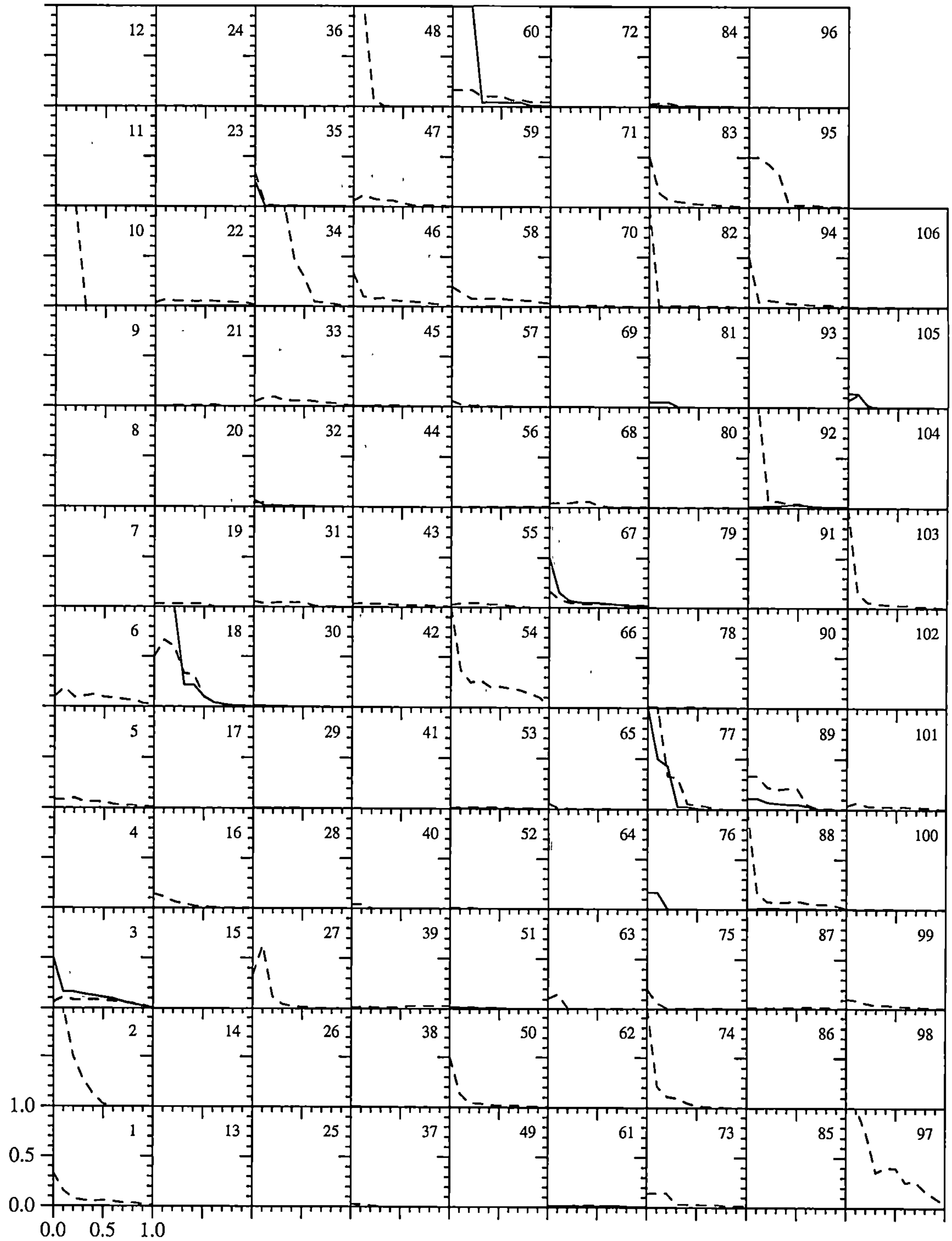

Figure B.2: Precision-recall graphs : $\vec{q_{add}}$ versus original LSI.

Table B.2: $P_{q_{\bar{a}dd}}$ versus P_{LSI}.

Query	P_{LSI}	P_{KELSI}	Query	P_{LSI}	P_{KELSI}	Query	P_{LSI}	P_{KELSI}
1	0.0863	0.0001	37	0.0065	0.0001	73	0.0567	0.0002
2	0.2657	0.0000	38	0.0033	0.0003	74	0.1388	0.0001
3	0.0714	**0.1338**	39	0.0303	0.0011	75	0.0280	0.0003
4	0.0001	0.0000	40	0.0081	0.0001	76	0.0318	0.0310
5	0.0557	0.0001	41	0.0010	0.0001	77	0.2584	0.1812
6	0.1007	0.0001	**42**	0.0002	**0.0002**	78	0.0073	0.0001
7	0.0010	0.0002	43	0.0261	0.0007	79	0.0007	0.0005
8	0.0000	0.0000	44	0.0007	0.0006	80	0.0025	0.0000
9	0.0002	0.0000	45	0.0065	0.0022	81	0.0143	0.0142
10	0.2734	0.0000	46	0.0821	0.0001	82	0.1001	0.0002
11	0.0046	**0.0048**	47	0.0485	0.0001	83	0.0847	0.0002
12	0.0000	0.0000	48	0.1894	0.0006	84	0.0170	0.0079
13	0.0003	0.0000	49	0.0000	0.0000	85	0.0001	0.0000
14	0.0012	0.0001	50	0.0754	0.0001	86	0.0000	0.0000
15	0.0001	0.0000	51	0.0089	0.0050	87	0.0167	0.0010
16	0.0388	0.0002	52	0.0036	0.0000	88	0.1557	0.0089
17	0.0003	0.0001	53	0.0120	0.0033	89	0.1456	0.0462
18	0.2584	**0.3298**	54	0.2684	0.0004	90	0.0006	0.0000
19	0.0176	0.0000	55	0.0248	0.0001	91	0.0003	0.0000
20	0.0006	0.0000	56	0.0075	0.0029	92	0.2077	0.0229
21	0.0085	0.0002	57	0.0114	0.0001	93	0.0000	0.0000
22	0.0511	0.0069	58	0.0876	0.0029	94	0.0796	0.0006
23	0.0001	0.0001	59	0.0003	0.0000	95	0.1703	0.0001
24	0.0015	0.0000	**60**	0.0995	**0.2954**	96	0.0009	0.0008
25	0.0001	0.0000	61	0.0161	0.0141	97	0.4390	0.0000
26	0.0011	0.0001	62	0.0005	0.0003	98	0.0003	0.0001
27	0.1348	0.0005	63	0.0295	0.0002	99	0.0424	0.0003
28	0.0000	0.0000	**64**	0.0007	**0.0009**	100	0.0036	0.0000
29	0.0044	0.0023	65	0.0085	0.0001	101	0.0318	0.0002
30	0.0049	**0.0051**	66	0.0005	0.0003	102	0.0007	0.0000
31	0.0302	0.0005	**67**	0.0590	**0.0911**	103	0.1261	0.0001
32	0.0135	0.0122	68	0.0319	0.0001	104	0.0010	0.0000
33	0.0613	0.0027	69	0.0019	0.0014	**105**	0.0262	**0.0275**
34	0.4432	0.0001	70	0.0108	0.0002	106	0.0038	0.0002
35	0.0383	0.0276	71	0.0004	0.0001			
36	0.0040	0.0000	72	0.0017	0.0001			

Table B.3: $P_{q_{\vec{proj}}}$ versus P_{LSI}.

Query	P_{LSI}	P_{KELSI}	Query	P_{LSI}	P_{KELSI}	Query	P_{LSI}	P_{KELSI}
1	0.0863	0.0785	**37**	0.0065	**0.0100**	73	0.0567	0.0419
2	0.2657	0.2346	38	0.0033	0.0027	74	0.1388	0.1012
3	0.0714	**0.0962**	39	0.0303	0.0284	75	0.0280	0.0140
4	0.0001	0.0001	**40**	0.0081	**0.0084**	76	0.0318	0.0175
5	0.0557	**0.0670**	**41**	0.0010	**0.0015**	**77**	0.2584	**0.2844**
6	0.1007	**0.1529**	42	0.0002	0.0002	78	0.0073	0.0066
7	0.0010	**0.0015**	**43**	0.0261	**0.1285**	**79**	0.0007	**0.0007**
8	0.0000	0.0000	**44**	0.0007	**0.0008**	**80**	0.0025	**0.0031**
9	0.0002	**0.0002**	**45**	0.0065	**0.0090**	81	0.0143	0.0106
10	0.2734	0.2733	46	0.0821	0.0581	82	0.1001	0.0997
11	0.0046	0.0045	**47**	0.0485	**0.0610**	83	0.0847	0.0731
12	0.0000	**0.0000**	**48**	0.1894	**0.2134**	84	0.0170	0.0105
13	0.0003	0.0002	49	0.0000	0.0000	85	0.0001	0.0000
14	0.0012	0.0011	**50**	0.0754	**0.1247**	86	0.0000	0.0000
15	0.0001	**0.0001**	51	0.0089	0.0074	87	0.0167	0.0128
16	0.0388	**0.0390**	**52**	0.0036	**0.0052**	**88**	0.1557	**0.1692**
17	0.0003	0.0002	53	0.0120	0.0060	89	0.1456	0.1267
18	0.2584	**0.2856**	**54**	0.2684	**0.3081**	**90**	0.0006	**0.0007**
19	0.0176	**0.0325**	55	0.0248	0.0159	91	0.0003	0.0003
20	0.0006	**0.0008**	56	0.0075	0.0038	**92**	0.2077	**0.2211**
21	0.0085	0.0079	**57**	0.0114	**0.0539**	93	0.0000	0.0000
22	0.0511	**0.0797**	**58**	0.0876	**0.1054**	94	0.0796	0.0793
23	0.0001	**0.0002**	**59**	0.0003	**0.0005**	95	0.1703	0.1441
24	0.0015	0.0015	**60**	0.0995	**0.1627**	96	0.0009	0.0009
25	0.0001	0.0001	61	0.0161	0.0082	97	0.4390	0.4386
26	0.0011	0.0011	**62**	0.0005	**0.0006**	98	0.0003	0.0001
27	0.1348	0.0993	63	0.0295	0.0265	**99**	0.0424	**0.1357**
28	0.0000	0.0000	64	0.0007	0.0007	100	0.0036	0.0029
29	0.0044	0.0039	65	0.0085	0.0041	**101**	0.0318	**0.0365**
30	0.0049	0.0045	66	0.0005	0.0004	**102**	0.0007	**0.0009**
31	0.0302	0.0293	**67**	0.0590	**0.1378**	103	0.1261	0.0680
32	0.0135	**0.0456**	68	0.0319	0.0127	104	0.0010	0.0009
33	0.0613	0.0537	69	0.0019	0.0017	**105**	0.0262	**0.0269**
34	0.4432	0.4396	**70**	0.0108	**0.0147**	**106**	0.0038	**0.0042**
35	0.0383	0.0276	71	0.0004	0.0004			
36	0.0040	0.0019	**72**	0.0017	**0.0025**			

Appendix C

Results of Query Matching Using MeSH Vectors

Results of query matching in Chapter 3, Section 3.2 are presented here. They include:

1. Precision-recall graphs for directly matched MeSH vectors versus the original LSI method.

2. Table of $P_{MeSH\ Direct\ Match}$ versus P_{LSI}.

3. Precision-recall graphs for two-step matched MeSH vectors versus the original LSI method.

4. Table of $P_{MeSH\ Two-Step\ Match}$ versus P_{LSI}.

5. Table of $P_{MeSH\ Regular\ Expression\ Match}$ versus P_{LSI}.

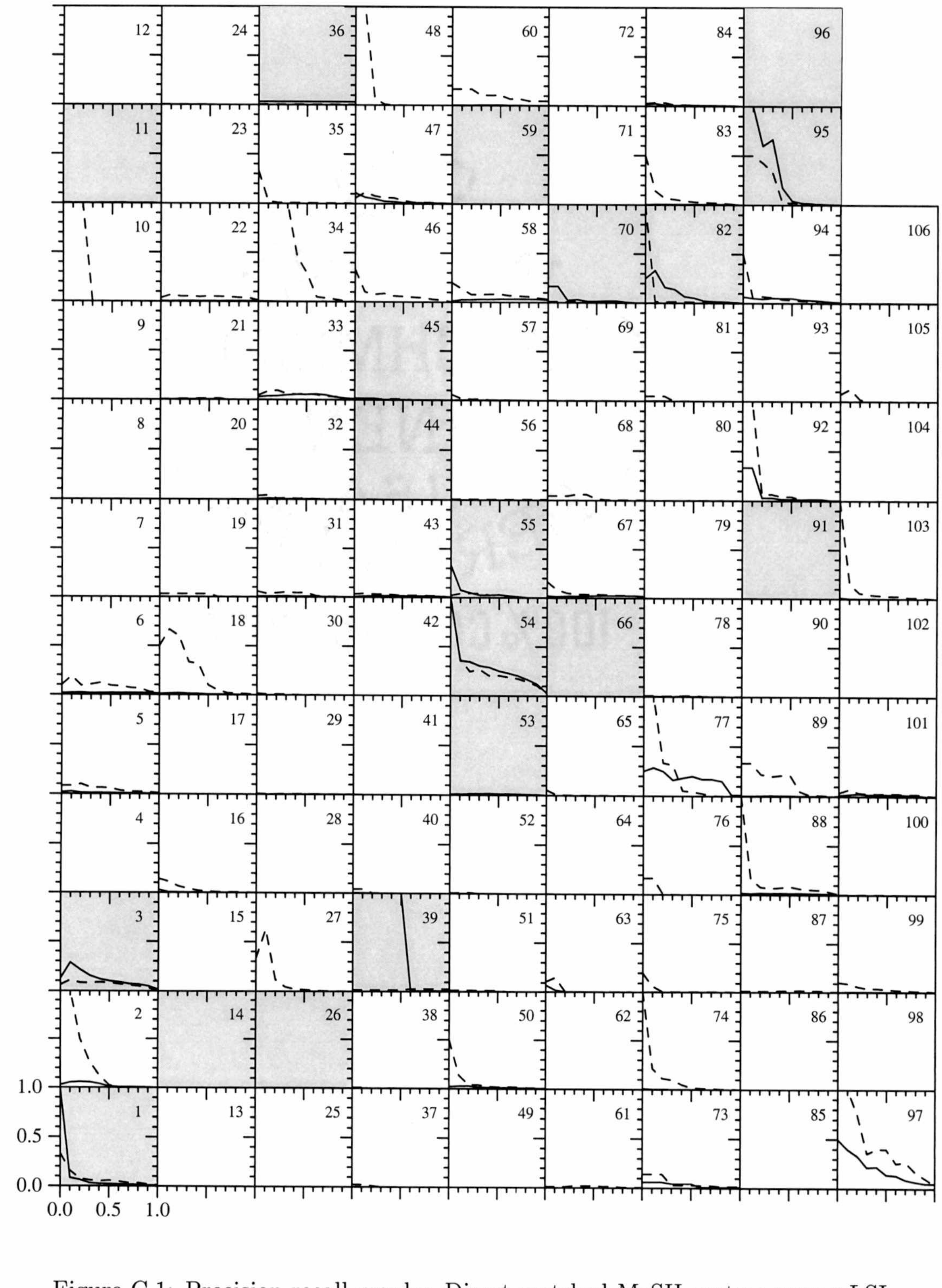

Figure C.1: Precision-recall graphs: Direct-matched MeSH vectors versus LSI.

Table C.1: $P_{MeSH\ Direct\ Match}$ versus P_{LSI} for 106 queries.

Query	P_{LSI}	P_{KELSI}	Query	P_{LSI}	P_{KELSI}	Query	P_{LSI}	P_{KELSI}
1	0.0863	**0.1187**	37	0.0065	0.0011	73	0.0567	0.0331
2	0.2657	0.0283	38	0.0033	0.0016	74	0.1388	0.0030
3	0.0714	**0.1354**	**39**	0.0303	**0.5492**	75	0.0280	0.0020
4	0.0001	0.0000	40	0.0081	0.0009	76	0.0318	0.0001
5	0.0557	0.0127	41	0.0010	0.0004	77	0.2584	0.1798
6	0.1007	0.0204	42	0.0002	0.0001	78	0.0073	0.0022
7	0.0010	0.0009	43	0.0261	0.0161	79	0.0007	0.0001
8	0.0000	0.0000	**44**	0.0007	**0.0007**	80	0.0025	0.0017
9	0.0002	0.0001	**45**	0.0065	**0.0067**	81	0.0143	0.0010
10	0.2734	0.0003	46	0.0821	0.0010	**82**	0.1001	**0.1136**
11	0.0046	**0.0049**	47	0.0485	0.0326	83	0.0847	0.0071
12	0.0000	0.0000	48	0.1894	0.0004	84	0.0170	0.0115
13	0.0003	0.0001	49	0.0000	0.0000	85	0.0001	0.0000
14	0.0012	**0.0012**	50	0.0754	0.0161	86	0.0000	0.0000
15	0.0001	0.0000	51	0.0089	0.0023	87	0.0167	0.0009
16	0.0388	0.0049	52	0.0036	0.0012	88	0.1557	0.0165
17	0.0003	0.0001	**53**	0.0120	**0.0130**	89	0.1456	0.0056
18	0.2584	0.0093	**54**	0.2684	**0.3024**	90	0.0006	0.0004
19	0.0176	0.0005	**55**	0.0248	**0.0536**	**91**	0.0003	**0.0006**
20	0.0006	0.0001	56	0.0075	0.0005	92	0.2077	0.0758
21	0.0085	0.0033	57	0.0114	0.0031	93	0.0000	0.0000
22	0.0511	0.0074	58	0.0876	0.0342	94	0.0796	0.0413
23	0.0001	0.0001	**59**	0.0003	**0.0017**	**95**	0.1703	**0.3285**
24	0.0015	0.0013	60	0.0995	0.0045	**96**	0.0009	**0.0012**
25	0.0001	0.0000	61	0.0161	0.0000	97	0.4390	0.2016
26	0.0011	**0.0015**	62	0.0005	0.0002	98	0.0003	0.0000
27	0.1348	0.0003	63	0.0295	0.0125	99	0.0424	0.0046
28	0.0000	0.0000	64	0.0007	0.0003	100	0.0036	0.0026
29	0.0044	0.0021	65	0.0085	0.0005	101	0.0318	0.0193
30	0.0049	0.0004	**66**	0.0005	**0.0006**	102	0.0007	0.0000
31	0.0302	0.0076	67	0.0590	0.0261	103	0.1261	0.0028
32	0.0135	0.0057	68	0.0319	0.0003	104	0.0010	0.0003
33	0.0613	0.0465	69	0.0019	0.0006	105	0.0262	0.0010
34	0.4432	0.0028	**70**	0.0108	**0.0446**	106	0.0038	0.0037
35	0.0383	0.0004	71	0.0004	0.0003			
36	0.0040	**0.0333**	72	0.0017	0.0002			

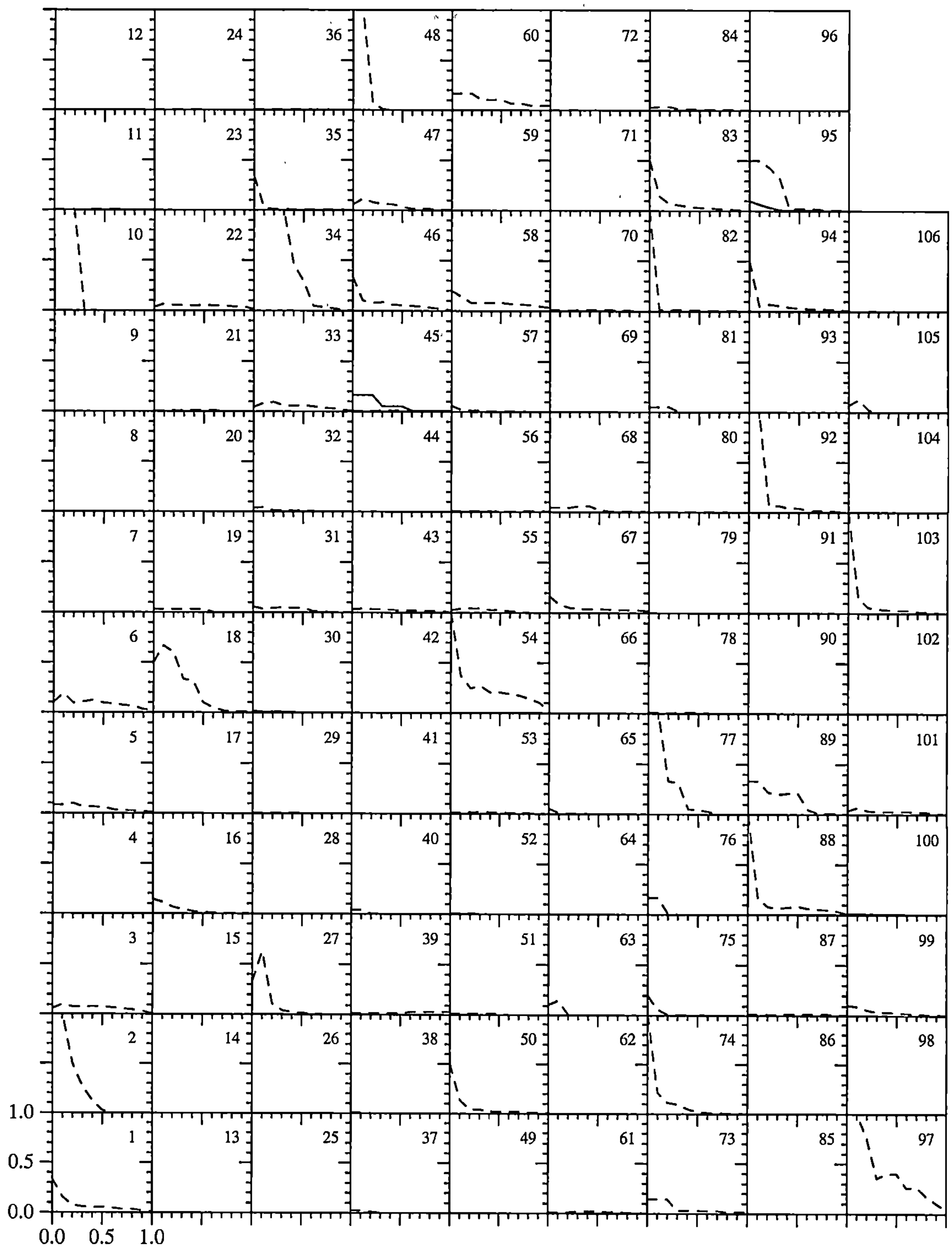

Figure C.2: Precision-recall graphs: Two-step matched MeSH vectors versus LSI.

Table C.2: $P_{MeSH\ Two-Step\ Match}$ versus P_{LSI} for 106 queries.

Query	P_{LSI}	P_{KELSI}	Query	P_{LSI}	P_{KELSI}	Query	P_{LSI}	P_{KELSI}
1	0.0863	0.0003	37	0.0065	0.0002	73	0.0567	0.0000
2	0.2657	0.0001	38	0.0033	0.0000	74	0.1388	0.0001
3	0.0714	0.0007	39	0.0303	0.0016	75	0.0280	0.0001
4	0.0001	0.0000	40	0.0081	0.0000	76	0.0318	0.0001
5	0.0557	0.0001	41	0.0010	0.0001	77	0.2584	0.0000
6	0.1007	0.0004	42	0.0002	0.0001	78	0.0073	0.0001
7	0.0010	0.0000	43	0.0261	0.0002	79	0.0007	0.0002
8	0.0000	0.0000	44	0.0007	0.0000	80	0.0025	0.0000
9	0.0002	0.0000	**45**	0.0065	**0.0643**	81	0.0143	0.0002
10	0.2734	0.0000	46	0.0821	0.0001	82	0.1001	0.0006
11	0.0046	0.0001	47	0.0485	0.0001	83	0.0847	0.0002
12	0.0000	0.0000	48	0.1894	0.0002	84	0.0170	0.0001
13	0.0003	0.0001	49	0.0000	0.0000	85	0.0001	0.0000
14	0.0012	0.0000	50	0.0754	0.0001	86	0.0000	0.0000
15	0.0001	0.0000	51	0.0089	0.0000	87	0.0167	0.0000
16	0.0388	0.0001	52	0.0036	0.0001	88	0.1557	0.0032
17	0.0003	0.0001	53	0.0120	0.0008	89	0.1456	0.0000
18	0.2584	0.0001	54	0.2684	0.0046	90	0.0006	0.0001
19	0.0176	0.0004	55	0.0248	0.0001	91	0.0003	0.0000
20	0.0006	0.0000	56	0.0075	0.0003	92	0.2077	0.0025
21	0.0085	0.0000	57	0.0114	0.0001	93	0.0000	0.0000
22	0.0511	0.0002	58	0.0876	0.0003	94	0.0796	0.0049
23	0.0001	0.0000	59	0.0003	0.0001	95	0.1703	0.0221
24	0.0015	0.0000	60	0.0995	0.0000	96	0.0009	0.0005
25	0.0001	0.0001	61	0.0161	0.0000	97	0.4390	0.0001
26	0.0011	0.0001	62	0.0005	0.0004	98	0.0003	0.0000
27	0.1348	0.0017	63	0.0295	0.0001	99	0.0424	0.0001
28	0.0000	0.0000	64	0.0007	0.0002	**100**	0.0036	**0.0074**
29	0.0044	0.0022	65	0.0085	0.0001	101	0.0318	0.0000
30	0.0049	0.0036	66	0.0005	0.0001	102	0.0007	0.0001
31	0.0302	0.0000	67	0.0590	0.0005	103	0.1261	0.0002
32	0.0135	0.0001	68	0.0319	0.0000	**104**	0.0010	**0.0016**
33	0.0613	0.0007	69	0.0019	0.0001	105	0.0262	0.0001
34	0.4432	0.0006	70	0.0108	0.0000	106	0.0038	0.0001
35	0.0383	0.0004	71	0.0004	0.0000			
36	0.0040	0.0000	72	0.0017	0.0001			

Table C.3: $P_{MeSH\ Regular\ Expression\ Match}$ versus P_{LSI} for 106 queries.

Query	P_{LSI}	P_{KELSI}	Query	P_{LSI}	P_{KELSI}	Query	P_{LSI}	P_{KELSI}
1	0.0863	0.0023	**37**	0.0065	**0.0659**	73	0.0567	0.0331
2	0.2657	0.0283	38	0.0033	0.0000	74	0.1388	0.0710
3	0.0714	0.0349	39	0.0303	0.0033	75	0.0280	0.0203
4	0.0001	0.0000	40	0.0081	0.0002	76	0.0318	0.0002
5	0.0557	0.0000	**41**	0.0010	**0.0109**	77	0.2584	0.1798
6	0.1007	**0.1514**	**42**	0.0002	**0.0479**	78	0.0073	0.0002
7	0.0010	0.0000	**43**	0.0261	**0.0603**	**79**	0.0007	**0.0029**
8	0.0000	0.0000	44	0.0007	0.0004	80	0.0025	0.0000
9	0.0002	**0.0002**	**45**	0.0065	**0.0643**	81	0.0143	0.0032
10	0.2734	0.1644	**46**	0.0821	**0.2724**	82	0.1001	0.0043
11	0.0046	**0.0049**	47	0.0485	0.0312	83	0.0847	0.0000
12	0.0000	0.0000	48	0.1894	0.0004	**84**	0.0170	**0.0377**
13	0.0003	0.0000	49	0.0000	0.0000	**85**	0.0001	**0.0079**
14	0.0012	**0.0012**	50	0.0754	0.0161	86	0.0000	0.0000
15	0.0001	**0.0125**	51	0.0089	0.0000	87	0.0167	0.0004
16	0.0388	**0.1234**	**52**	0.0036	**0.0100**	88	0.1557	0.0165
17	0.0003	**0.0445**	**53**	0.0120	**0.0760**	89	0.1456	0.0013
18	0.2584	0.0021	**54**	0.2684	**0.3024**	90	0.0006	0.0004
19	0.0176	0.0002	**55**	0.0248	**0.1382**	91	0.0003	0.0000
20	0.0006	0.0001	56	0.0075	0.0006	92	0.2077	0.0000
21	0.0085	0.0033	57	0.0114	0.0024	93	0.0000	0.0000
22	0.0511	0.0135	58	0.0876	0.0000	94	0.0796	0.0413
23	0.0001	**0.0082**	59	0.0003	0.0000	**95**	0.1703	**0.3285**
24	0.0015	0.0003	60	0.0995	0.0002	96	0.0009	0.0001
25	0.0001	**0.0398**	61	0.0161	0.0000	97	0.4390	0.0000
26	0.0011	**0.0581**	**62**	0.0005	**0.0921**	98	0.0003	0.0000
27	0.1348	0.0000	63	0.0295	0.0057	**99**	0.0424	**0.1071**
28	0.0000	0.0000	**64**	0.0007	**0.2290**	100	0.0036	0.0000
29	0.0044	**0.0161**	65	0.0085	0.0000	101	0.0318	0.0012
30	0.0049	0.0004	**66**	0.0005	**0.0006**	**102**	0.0007	**0.0859**
31	0.0302	0.0011	67	0.0590	0.0261	103	0.1261	0.0079
32	0.0135	0.0057	68	0.0319	0.0051	**104**	0.0010	**0.0013**
33	0.0613	0.0000	**69**	0.0019	**0.0766**	105	0.0262	0.0010
34	0.4432	0.0000	**70**	0.0108	**0.0446**	106	0.0038	0.0031
35	0.0383	0.0000	**71**	0.0004	**0.0054**			
36	0.0040	0.0001	**72**	0.0017	**0.0950**			

Vita

David Guo was born in Beijing, China on May 4, 1971. He immigrated to the United States in 1988. He received a Bachelor of Science degree in Geology from Rensselaer Polytechnic Institute in 1993. In January 1999, he entered the Computer Science graduate program at the University of Tennessee in Knoxville, Tennessee and received a Master of Science degree in Computer Science in May, 2001.