2015

Ranking components of scientific software using spectral methods

Khan, Soma Farin

Lethbridge, Alta : University of Lethbridge, Dept. of Mathematics and Computer Science

RANKING COMPONENTS OF SCIENTIFIC SOFTWARE USING SPECTRAL METHODS

SOMA FARIN KHAN
Bachelor of Science, North South University, 2013

A Thesis
Submitted to the School of Graduate Studies
of the University of Lethbridge
in Partial Fulfillment of the
Requirements for the Degree

MASTER OF SCIENCE

Department of Mathematics and Computer Science
University of Lethbridge
LETHBRIDGE, ALBERTA, CANADA

RANKING COMPONENTS OF SCIENTIFIC SOFTWARE USING SPECTRAL METHODS

SOMA FARIN KHAN

Date of Defense: 7th October, 2015

Dr. Shahadat Hossain Supervisor	Associate Professor	Ph.D.
Dr. Daya Gaur Committee Member	Professor	Ph.D.
Dr. Robert Benkoczi Committee Member	Associate Professor	Ph.D.
Dr. Howard Cheng Chair, Thesis Examination Committee	Associate Professor	Ph.D.

Dedication

To

my parents.

Abstract

In this thesis we explore the centrality rankings of functions in call graphs of scientific software using spectral method. Dependency Structure Matrix (DSM) is used as a modeling tool to represent and examine pattern of inter-dependencies among functions. We compute the hubs and authorities in directed networks using functions of matrices. The non-symmetry nature of the dependency relations is addressed by bipartization, i.e., by defining a symmetric matrix B using the original matrix and its transpose as shown below:

$$B = \begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix}$$

We use the matrix exponential method for computing hubs and authorities. We show that the hub and authority ranking provided by the diagonal entries of the matrix exponential may vary from the ranking provided using HITS algorithm. These two methods have been applied on both non-weighted and weighted call graphs of three scientific software and the results have been analyzed.

Acknowledgments

In full gratitude I would like to acknowledge the following individuals who encouraged, inspired and supported me by sacrificing their valuable time.

Credit for much of the work described in this thesis belongs to my Supervisor, Dr. Shahadat Hossain, for his insight, guidance, and patience. He provided for an excellent research environment, left me enough freedom to do things the way I thought they should be done, and was always available to discuss ideas and problems.

I also want to take a moment to thank my supervisory committee members, Dr Daya Guar and Dr Robert Benkoczi.

I must acknowledge four of my fellow graduates, Rumana Q Tithi, Anik Saha, Mahmudun Nabi and S M Erfanul Kabir, who have shared their precious time to help me with my work. They have guided me to the right direction by giving their valuable suggestions.

Finally, whose inspiration and support lead me to move forward is my family. I acknowledge each and everybody of my family for their cordial support. I would like to throw a big thank to all of them.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In this thesis we study software systems specifically designed for problems arising in scientific and engineering applications [15]. There are still some significant queries, regarding the design structure of scientific software, that need to be addressed. The architecture of a software system involves the specification of the elements it is composed of and the pattern of interactions among those elements. It is widely accepted that a formal and sound basis for the representation of software architecture is beneficial for the effective development and maintenance of software systems.

Network science is an emerging approach for modeling and analyzing large and complex systems that arise in diverse scientific disciplines from social network analysis to epidemiological studies [7] [3] [4] [5].

A system is a combination of interacting elements organized to achieve one or more stated purposes [6]. A network is an abstraction of objects called components (of a system) and their interactions. Network science provides us with algorithmic tools and techniques to study pattern of interactions between individual components to obtain valuable insights about the system under study.

In this thesis, we study the architecture of scientific software represented as a component dependence network. The central research question that we address in this thesis is what are the important components in the network. In the language of network science the notion of importance of a component is given by its centrality in the network. We employ spectral techniques to analyze the component dependence network of a selected collection of scientific software. We present and analyze results from numerical testing.

1.1 Motivation and background

In this thesis, we employ a set of quantitative measures to identify important design elements in scientific software by analyzing the interactions between them. Scientific software systems are complex products varying from thousands to millions of lines of source code, embodying implicit design decisions, internal and external constraints, different technical and non-technical concerns [13]. In our work, the centrality analysis of call graphs has been extended beyond nodal degree. More precisely, the importance of a node in a call graph is determined by its ”connectedness”. As a caller, a function's importance or centrality is determined by the importance of the functions that it calls. Likewise, as a callee, a function is considered important if it is called by important functions. Therefore the centrality measure captures information beyond what is provided by simple in- or out-degree count. According to Gilbert Strang [25], for a directed graph if each pair of vertices are connected by a directed path and if its adjacency matrix is irreducible then according to Perron-Frobenius [23], the result states that the largest eigenvalue of the matrix is positive and simple.The associated eigenvector has non-negative components and their values represent relative importance of the respective nodes. We interpret and compare different eigenvector centrality measures applicable to function call graphs. The dependency structure matrix (DSM) has been used as a tool to represent, analyze and compare structural metrics. DSM is a network modeling tool used to represent the elements comprising a system and their interactions [1]. It is a highly flexible system modeling tool. The DSM has been modified and extended by many researchers and practitioners since its initial development.
DSM allows us to apply linear algebra techniques to analyze dependency network. DSM can meaningfully represent a fairly large, complex system in a relatively small space.

In this work, we are trying to determine the importance or centrality of the components

of a scientific software which will be helpful in understanding several other things. For example, if there is a change made in one component, what are the impacts of that change in the other components. Another example would be if a network is redesigned, which of the components will be affected the most. Instead of using degree-based centrality we are using spectral method for the analysis. It is expected for a spectral method to give more accurate information about centrality.

1.1.1 Scientific computing software

According to Marco A Gonzela [13], software systems are complex products varying from thousands to millions of lines of source code, embodying implicit design decisions, internal and external constraints, different technical and non-technical concerns. Though a number of scientific computing software applications have been developed as a proof-of-concept tool, powerful hardware resources facilitated scientific software to solve and simulate large problem. With the rise of more powerful hardware resources and super computers, an increasing number of scientific applications are being developed to carry out wide-ranging simulation runs which were previously not possible [16]. Unlike the one-time throwaway computer code, these simulation software applications are highly complex and extensive (IPSL-CM5, 2011) containing millions of lines of computer code. The applications involve substantial investment in time and other computational and manpower resources, and tend to have life-cycles measured in tens of years. The various importance of scientific software are re-usability, extensibility, efficiency, portability, correctness, robustness and ease of use.

Software systems are composed of one or more independently developed modules. Each module is a segment of the software. When software module A uses another software module B then we say module A is dependent on module B. Software dependency can be static or dynamic. The concept of static dependency is that one module is required to compile another module. Dynamic dependency, also known as run-time dependency, is a record of

an execution of the program. In our work dynamic dependency is not appropriate as not all subroutines are executed at run time, so complete code dependency will be missing to some extent.

1.1.2 Software architecture

Software architecture helps to understand the complexity of large software systems. It is the high level structure of the software system. Software architecture is an ”intellectually graspable” abstraction of a complex system. This abstraction provides a number of benefits. For example:

(i) Even before the system is built, the software architecture gives a basis for analysis of software systems' behavior.

(ii) It provides a basis for re-use of elements and decisions. A complete software architecture or parts of it can be re-used across multiple systems whose stakeholders require similar quality attributes or functionality. It saves design costs.

(iii) It supports early design decisions that impact a system's development, deployment, and maintenance life.

1.2 Dependency relationship

Dependency relationship can be viewed as a complex network. This complex network view has been successfully applied in numerous areas. For example, nature's most complex system, the human cerebral cortex has been used in the paper [5]. The initial success of a new weighted network communicability measure in distinguishing local and global differences between diseased patients and controls is reported.

1.3 Organization of the thesis

The rest of this thesis is organized as follows. In Chapter 2, we discuss the description of the dependency extraction and Modeling. In this Chapter we discuss about the different tools that are used to extract and visualize call graphs. And finally says about the tool that was used for data extraction in our thesis.

Chapter 3 presents detailed description about component centrality using spectral methods. Numerical experiments are shown in chapter 4. The results have been discussed and compared.

Finally, conclusions drawn from our experiments are discussed in Chapter five. Future work has also been added in this Chapter.

1.4 Contribution of the thesis

In this thesis, the centrality analysis of call graphs has been extended beyond nodal degree. Our main objective is to determine the importance or centrality of the components of a scientific software which will be helpful in understanding several other things. For example, if there is a change made in one component, what are the impacts of that change in the other components. Another example would be if a network is redesigned, which of the components will be affected the most. Instead of using degree-based centrality we have used spectral method for the analysis. More specifically the thesis addresses the following issues.

- We extend the notion of centrality of software components, as discussed in [30] [17], beyond nodal degrees. The spectral techniques employed have yields more accurate rankings as the methods are based on the entire network topology.
- The ranking methods developed in the paper are directly applicable to engineering systems design applications. Smith and Eppinger [24] applied DSM model to a Brake-system Design. The objective of the analysis was to rank the subsystems

in the design space with regard to the convergence of design iteration. The spectral method employed by Smith and Eppinger can be extended to centrality methods HITS and matrix exponential.

- The spectral ranking methods will be beneficial in analyzing legacy software where there are a little or no documentation on the code is available. The spectral techniques can be applied to identify software components that are reusable.

- To the best of our knowledge, the work presented in the thesis is the first time that spectral analysis has been applied to scientific software domain.

A part of the work presented in this thesis has been accepted for publication as a proceeding paper in the 17th International Dependency And Structural Modeling Conference, DSM 2015, Fort Worth, Texas, USA.

Title : On Ranking Components in Scientific Software

Author(s) : Hossain, Shahadat; Khan, Soma Farin; Quashem, Rumana

Chapter 2

Dependency Extraction and Modeling

2.1 Design Structure Matrix

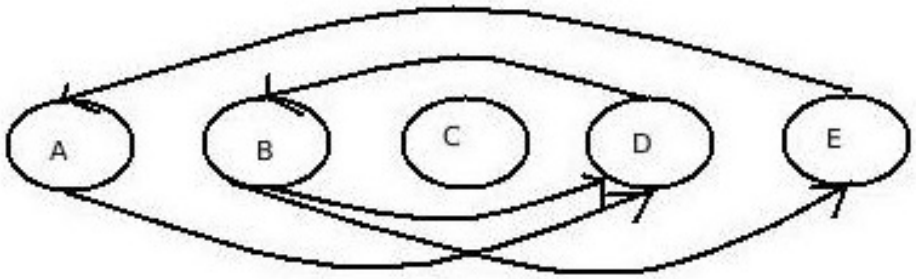
Complexity of product design has been a critical topic for both researchers and managers for many years. However, with the help of a reasonable model, it has become possible to explore approaches to understand the complexity of the product. This model known as "Design Structure Matrix (DSM)" was originated by Don Steward in 1981 [19] for understanding interactions between product design elements. It is a tool used to compactly represent different types of dependencies that exist between different types of components. The dependency between one element and other is indicated with an off diagonal mark. The DSM is a powerful modeling tool that enables visualization of a system architecture and their interactions. System architecture is the structure of a system that includes components relationships to each other and the principle guiding its design and evolution.

An example of simple DSM is shown in Figure 2.1. The five system elements are labeled A to E. The mark in the off-diagonal cells represents an interaction. For example, reading across row D, we see that the element D has inputs from element A and B. Reading down the column D, we see that D has output going to element B. In this thesis, we used DSM to study the architectural properties of scientific software. It is used as a modeling tool to capture and analyze pattern of inter-dependencies among functions.

DSM, as a modeling tool, offers some salient advantages. It is a highly flexible system modeling tool. It allows us to apply linear algebra techniques to analyze dependency network. DSM can meaningfully represent a fairly large, complex system in a relatively small space. Graphs are used to represent the exactly same information that is in the DSM. But

	A	B	C	D	E
A	-				X
B		-		X	
C			-		
D	X	X		-	
E		X			-

(a)



(b)

Figure 2.1: (a) DSM (b) Its equivalent in diagraph form

graphs can be difficult to understand when the number of nodes and edges grow. The structured arrangement of elements and interactions provides a compact representation format.

2.2 Dependency Extraction

Examining program dependencies is challenging for large systems [26]. A number of tools exist for extracting and visualizing call graphs for facilitating software engineers to understand the program [22]. The two main classes of call graph extractor are identified by Telea et al [26]. Lightweight and Heavyweight are the two main classes of the call

graph extractors. Lightweight extraction provides a fraction of the entire static information whereas Heavyweight extractors provide nearly a complete call graph by performing full parsing and type checking. Strict and tolerant are the two types of Heavyweight extractors. Strict heavyweight extractors are based on compiler which stops when there is a lexical or syntax error. However, tolerant ones are based on Fuzzy or Generalized Left Reduced (GLR) parsing. GLR parsing is one of the most efficient parsing for context free grammer. A Tolerant heavyweight extractor provides complete static call graphs.The tool OINK is a tolerant heavyweight extractor. There exists another tool, called Lattix LDM. In contrast to the more common dependency analysis tools that use box-and-line diagrams to show the dependencies among components, this tool uses a DSM.

In [30], the OINK-based call-and-structure extractor was used. As a result, only file and call graphs could be extracted.

The tool that has been used for data extractions in this thesis is *Understand* [29] which has more choice of dependency extraction.

Understand is used to analyze the dependencies between software artifacts in a project. The application supports a wide range of programming languages, including Ada, C++, FORTRAN, Java, JOVIAL and Delphi/Pascal.

It offers code navigation using a detailed cross-referencing, a syntax-colorizing ”smart” editor, and a variety of graphical reverse engineering views. Understand has architecture features that help us create hierarchical aggregations of source code units. We can name these units and manipulate them in various ways to create interesting hierarchies for analysis.

The Dependency Browser helps to examine which items are dependent on others. The Dependency Browser can be used with architecture nodes, files, classes, packages, and interfaces. With the help of the dependency browser different types of dependencies can be extracted. A brief explanation of the dependencies that can be extracted using *Understand* is given below. We have used a small project to show the dependencies.

Call Dependency:

Figure 2.3 shows all the call dependencies of a Scanner project. The files with the outgoing edges are dependent on the files with the incoming edges. For example, by looking at Figure 2.3 we can tell that the file symtable.cc is dependent on the file token.cc and token.h, as there is an edge from symtable.cc to token.h and token.cc. The integer value on the edge represents the total number of dependencies. For instance, there are total 11 function calls from symtable.cc to token.cc. The details of these calls or dependencies can be found in the *Information Browser*.

The 11 call dependencies are:

1) search() calls getLexeme() 4 times in symtable.cc (line 44,49,54,56)

2) insert() calls getLexeme() 4 times in symtable.cc (line 93,114,118,133)

3) insert() calls token() 4 times in symtable.cc (line 105,122,136)

Figure 2.2 shows the two functions, Search() and Insert(), and the calls that have been made within those functions.

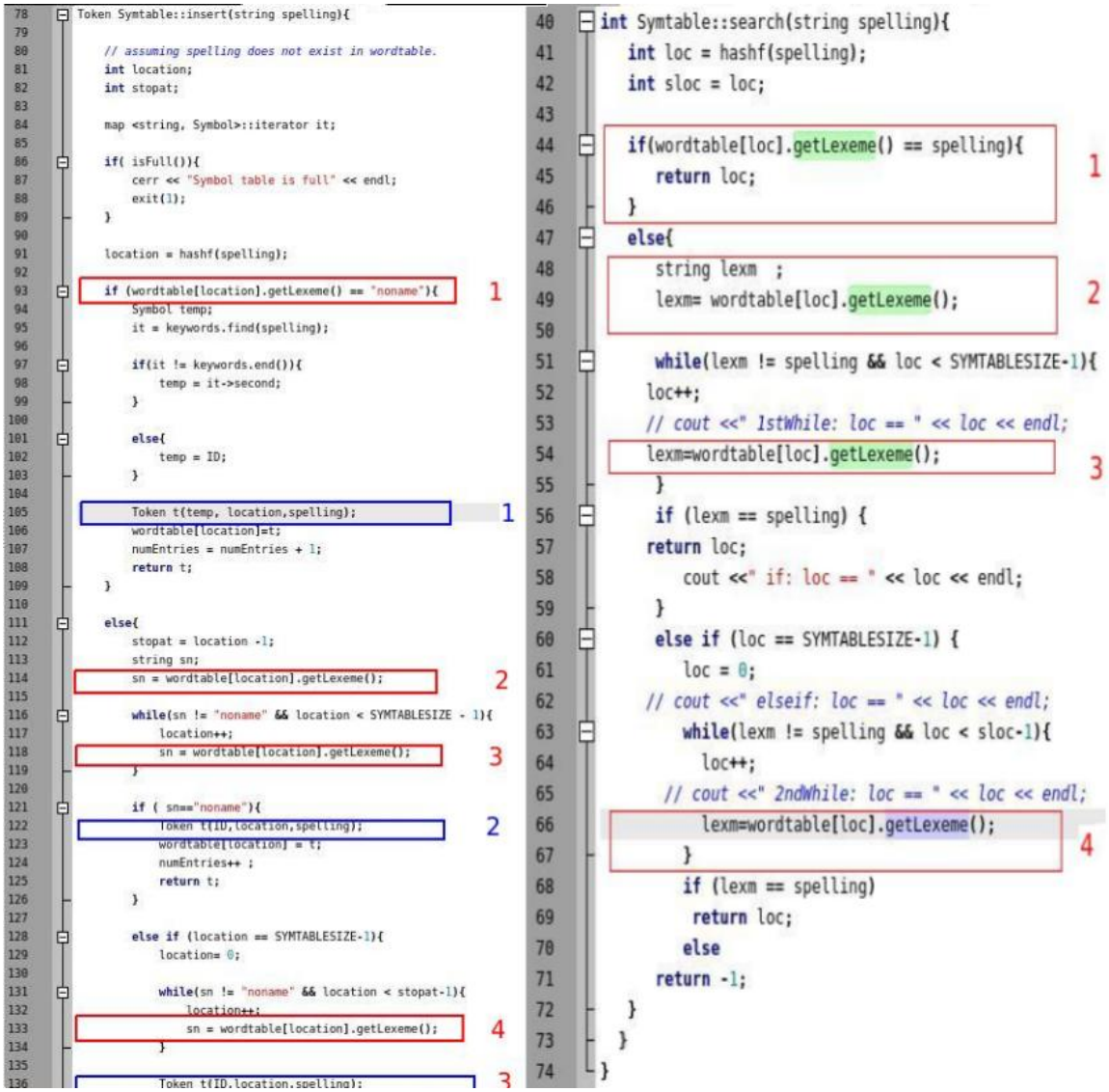
Uses Dependency:

The Uses Dependency Graph shows the various *uses* between two files. For example by looking at the use dependency graph we can tell that token.cc uses token.h 5 times. Figure 3 shows the Uses Dependency Graph. The edge from token.cc to token.h with the integer value 5 shows that token.cc uses 5 entities or objects from token.h. When we look at it closer we can see the following:

1) Token::getLexeme() returns svalue at token.cc

```
26
27  string Token::getLexeme(){
28      return svalue.lexeme;
29  }
```

2) Token::getLexeme() returns lexeme at token.cc



```
78  Token Symtable::insert(string spelling){
79
80      // assuming spelling does not exist in wordtable.
81      int location;
82      int stopat;
83
84      map <string, Symbol>::iterator it;
85
86      if( isFull()){
87          cerr << "Symbol table is full" << endl;
88          exit(1);
89      }
90
91      location = hashf(spelling);
92
93      if (wordtable[location].getLexeme() == "noname"){
94          Symbol temp;
95          it = keywords.find(spelling);
96
97          if(it != keywords.end()){
98              temp = it->second;
99          }
100
101         else{
102             temp = ID;
103         }
104
105         Token t(temp, location,spelling);
106         wordtable[location]=t;
107         numEntries = numEntries + 1;
108         return t;
109     }
110
111     else{
112         stopat = location -1;
113         string sn;
114         sn = wordtable[location].getLexeme();
115
116         while(sn != "noname" && location < SYMTABLESIZE - 1){
117             location++;
118             sn = wordtable[location].getLexeme();
119         }
120
121         if ( sn=="noname"){
122             Token t(ID,location,spelling);
123             wordtable[location] = t;
124             numEntries++ ;
125             return t;
126         }
127
128         else if (location == SYMTABLESIZE-1){
129             location= 0;
130
131             while(sn != "noname" && location < stopat-1){
132                 location++;
133                 sn = wordtable[location].getLexeme();
134             }
135
136             Token t(ID,location,spelling);
```

1 2 1 3 2 4 3

```
40  int Symtable::search(string spelling){
41      int loc = hashf(spelling);
42      int sloc = loc;
43
44      if(wordtable[loc].getLexeme() == spelling){
45          return loc;
46      }
47      else{
48          string lexm  ;
49          lexm= wordtable[loc].getLexeme();
50
51          while(lexm != spelling && loc < SYMTABLESIZE-1){
52          loc++;
53          // cout <<" 1stWhile: loc == " << loc << endl;
54          lexm=wordtable[loc].getLexeme();
55          }
56          if (lexm == spelling) {
57          return loc;
58              cout <<" if: loc == " << loc << endl;
59          }
60          else if (loc == SYMTABLESIZE-1) {
61              loc = 0;
62          // cout <<" elseif: loc == " << loc << endl;
63              while(lexm != spelling && loc < sloc-1){
64                loc++;
65           // cout <<" 2ndWhile: loc == " << loc << endl;
66                lexm=wordtable[loc].getLexeme();
67              }
68              if (lexm == spelling)
69               return loc;
70              else
71          return -1;
72      }
73    }
74  }
```

1 2 3 4

Figure 2.2: Two functions : search() and insert()

```
27  string Token::getLexeme(){
28      return svalue.lexeme;
```

3) Token::getSymbol() returns Token:sname at token.cc

```
19  Symbol  Token::getSymbol( ){
20      return sname;
```

4) Token::getValue() returns Token:svalue

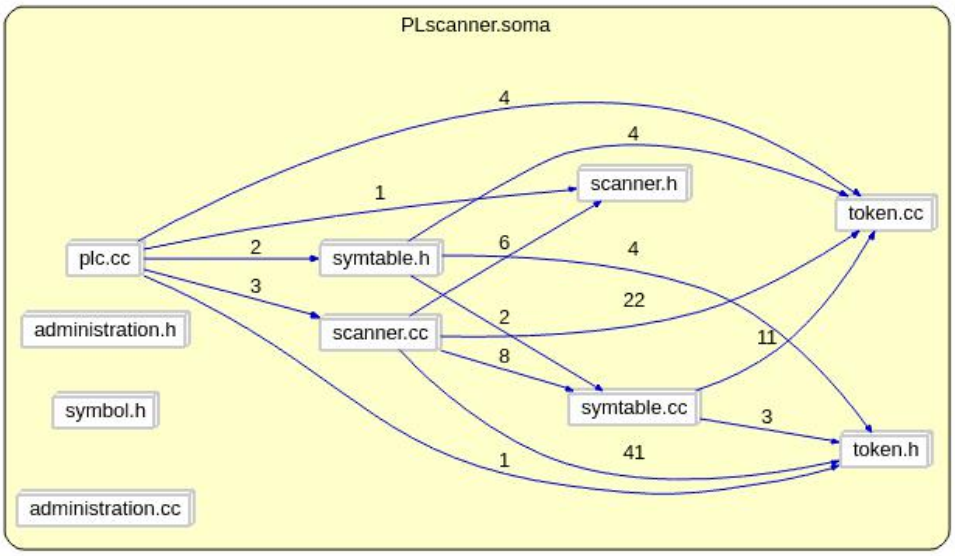


Figure 2.3: Call dependency

```
23  int Token::getValue(){
24      return svalue.value;
```

5) Token::getValue() returns Token:value

```
23  int Token::getValue(){
24      return svalue.value;
```

Thus making 5 Uses Dependency. These information are shown clearly on the Dependency Browser. The information of the dependency browser is saved in a comma separated value (csv) file called usesdb.csv. The csv file contains 4 columns in this case. To File represents the file name along with its path. References, From Entities, To Entities represents the total number of uses , number of function that uses other functions or objects, number of functions or objects that are being used by other functions respectively.

Init Dependency:

The init dependency focuses on the initialization of an object. In Figure 2.5 we can see that there is only one edge from token.cc to token.h with an integer value of 2. The picture

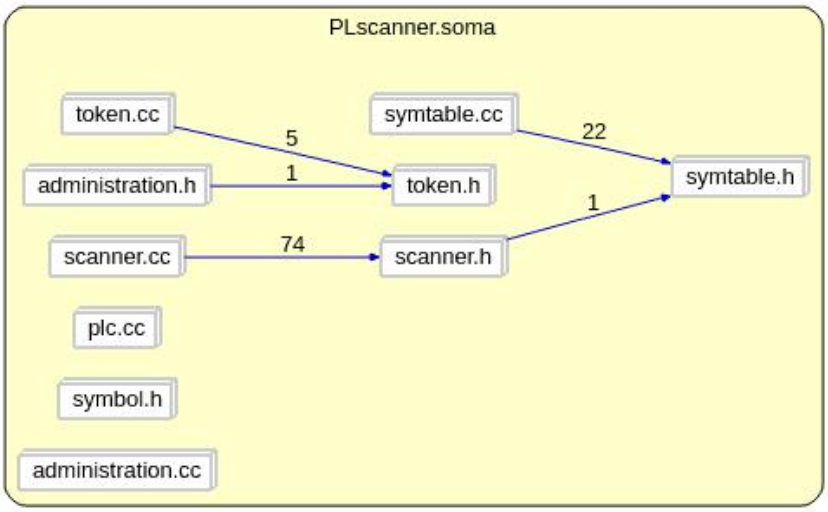


Figure 2.4: Uses dependency

below gives a closer view of the initialization of an object.

```
 9  Token::Token(){
10      sname = NONAME;
11      svalue.value = -1;
12      svalue.lexeme = "noname";
13  }
14
15  Token::Token(Symbol s, int v, string l){
16      sname = s; svalue.value = v; svalue.lexeme = l;
```

There are 2 initialization. Both are Token constructor (Token()::Token()) initializing *svalue*.

Set Dependency:

The set dependencies are shown in Figure 2.6. There are 12 set dependencies. The default constructor, Token, Sets the object *sname* 3 times at token.cc(line 10,11,12). Token Sets *lexeme* at token.cc(line 12) and Token Sets the object *value* at token.cc(line 11). Similarly, the other constructor also sets objects. Thus making 12 set dependencies.

Figure 2.5: Init dependency

```
8
9   Token::Token(){
10      sname = NONAME;          Token() sets sname at token.cc
11      svalue.value = -1;       Token() sets svalue at toke.cc . Token() sets value at token.cc
12      svalue.lexeme = "noname"; Token() sets svalue. Token() sets lexeme
13  }
14
15  Token::Token(Symbol s, int v, string l){
16      sname = s; svalue.value = v; svalue.lexeme = l;   // Token() sets sname,svalue,
17  }                                                     // value,lexeme at token.cc
```

Include Dependency

Figure 2.7 portrays the files with include dependency. The include Dependency graph shows the files that are needed to be included for implementation. For example, in Figure 2.7 we can see that symtable.h is dependent on token.h. It means that symtable.h has included the file token.h.

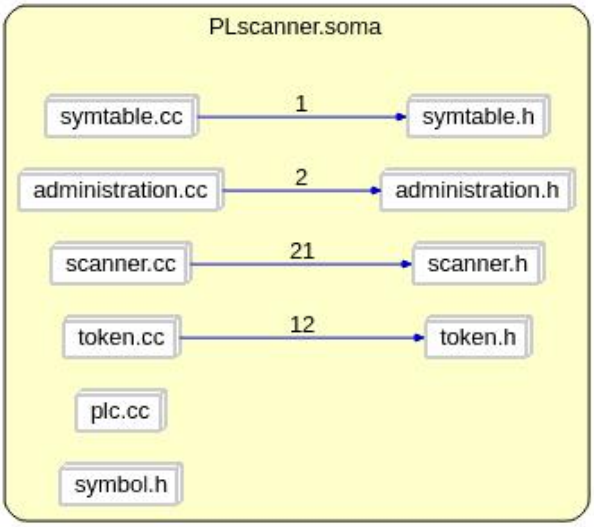


Figure 2.6: Set dependency graph

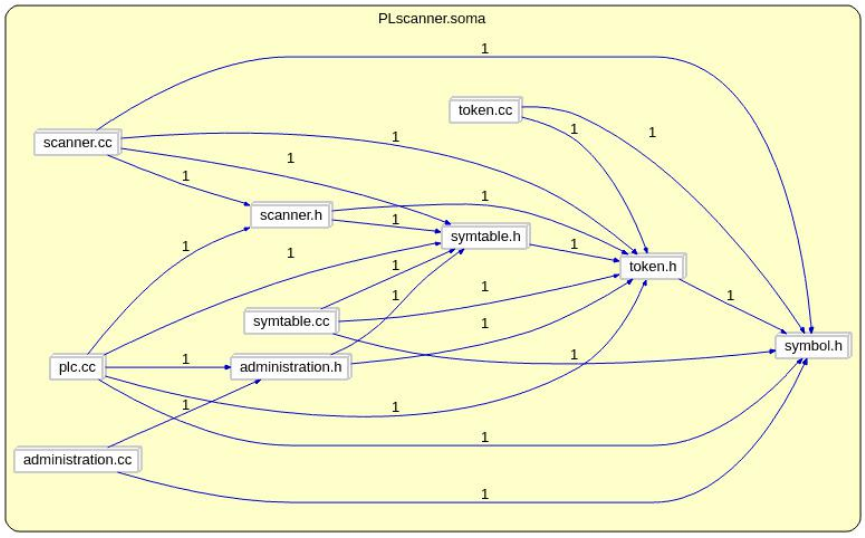


Figure 2.7: Include dependency graph

2.3 Matrices and Graphs

An English mathematician named James Josheph Sylvester was the first to introduce the term *Graph* in a paper published in 1878 in an interdisciplinary scientific journal called Nature, where he draws the correlation between ”quantic invariants” and ”co-variants” of algebra and molecular diagrams. Graphs have been used in an extensive range of applica-

tions since then.

In graph theory, a graph $G = (V,E)$ consists of two sets V and E . The set V is a finite set of vertices and the set E represents the pairwise relationship between the vertices in V. This pairwise relationship is called *edge*. For each edge $e = (u,v)$ where the two end points $u,v \in V$ are said to be *neighbors* or *adjacents*.

A *weighted graph* is a graph where a number is associated with an edge. These numbers are called *weight* or *cost* of the edge [20].

Graphs can be directed or undirected. In a *directed graph* the vertices are connected using an arrow like edge. An edge $e = (u,v)$ in a directed graph, u is the tail and v is the head of the edge. As a result, the pairs (u,v) and (v,u) represent two different edges in the graph. Directed graphs are also called *digraphs*. On the other hand, in an *undirected graph* the vertices $u,v \in e$ are un-ordered. Therefore, the pair (u,v) and (v,u) represents the same edge. Figure 2.8 and Figure 2.9 displays an undirected graph and directed graph respectively. *Subgraph* G' of G is a graph such that $V(G') \subseteq V(G)$ and $E(G') \subseteq E(G)$ where

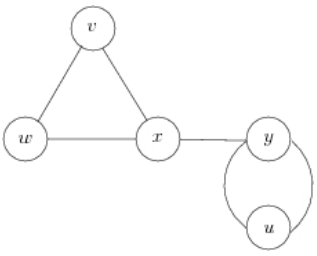


Figure 2.8: Undirected graph

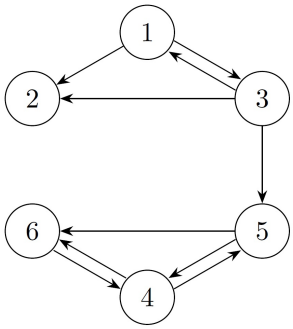


Figure 2.9: Directed graph

$E(G') = (u,v) \in E(G) | u,v \in V(G')$. A *path* in a graph is a sequence of edges that connect sequence of distinct vertices. For example, 1,3,2 is a path in Figure 2.9. The sequence of vertices are 1,3 and 2, and the sequence of edges are (1,3) and (3,2).

The *length* of a path is the number of edges in the path. A path is *simple* if all the vertices in the path are distinct.

A *cycle* is a path where the first and last vertices are the same. A *tree* is a connected acyclic

graph . If there is an edge (u,v) then *u* is said to be the *parent* of *v* and *v* is said to be the *child* of *u*. *Directed Acyclic Graph* is a directed graph which does not contain any cycles. The *degree* of a vertex *v* is the number of edges incident to that vertex *v* in a graph $G = (V, E)$. For a directed graph, the number of incoming edges are known as *in-degree* and the number of outgoing edges are known as *out-degree*.

A DSM is used to represent the exactly same information that is in the graph. The reason of using two different ways, graph and DSM, to represent the same information is because there is a trade-off. Graphs are more intuitive than DSM but they can be difficult to understand when the number of nodes and edges grow. A few dozens nodes can be enough to produce a graph too complex. On the other hand, large and complex graphs can be very efficiently represented by a DSM. In this thesis, we use the duality of a graph and

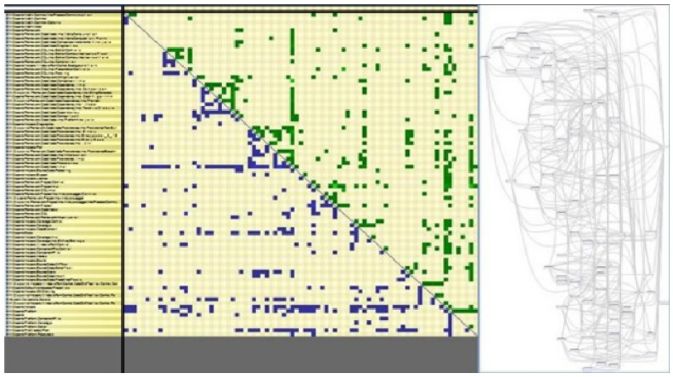


Figure 2.10: Large and complex graph efficiently represented by a DSM

matrix to effectively represent and compute quantitative information about the architecture of a software system using *Graph Theory* and *Linear Algebra*. Technically, the differences between matrix and graphs representations are minimal because data can be stored and presented in a variety of ways [21]. However, in practical applications in which software tools are being created to support new design methods, it has been found that the choice between using a matrix-based or graph-based representation has implications for the way the product information is captured, evaluated, and used [28].

Common product representation models can be generally classified as being matrix-based or graph-based model. Two such representations are High Definition Design Structure Matrix (HDDSM) and Component Flow Graph (CFG). The HDDSM modeling process reduces the effort required from the examiners and allows the creation of the model to be distributed across many people or completed at different points in time. A hierarchical system approach is used in which, at the beginning of the model creation process, the system is divided into groups. The system model is created using each group as a system element. Then for each group, the HDDSM can be created separately and merged back into the system model. After merging the group HDDSM into the system HDDSM, only a subset of interactions between elements of the group and other elements of the system needs to be reviewed [28].

The original intent of the CFG was to formally represent concepts derived from a function structure. In engineering design, the term *function* depicts the intended input/output relationship of a system whose purpose is to perform a task. If the overall task can be adequately defined, then it is possible to integrate input/outputs of all the quantities involves in an overall function. An overall function can usually be divided into identifiable sub-functions. The meaningful and compatible combination of sub-functions into an overall function produces a so called function structure. For details we refer to [11]. It has been established in Andrew's et al. paper that the formalized structure and systematic process of the HDDSM matrix-based model is found to be extremely valuable during capture phase and for mathematical evaluation, whereas, the CFG graph-based model strongly supports automated model transformation and intuitive interpretation for the user [28].

Chapter 3

Spectral Analysis of Dependencies

In this thesis our objective is to determine the centrality of the components of a scientific software. Instead of using degree-based centrality we are using spectral method for the analysis. Spectral methods rely on the eigenvalues of matrix representations of networks, and capture global information on structure. The basic premise is that networks have distinct spectra and hence the spectra are 'fingerprints' of network topology [9].
In this Chapter we will discuss about the two methods, namely HITS and Benzi et al. method, that we have applied to analyze the relative importance of components in scientific software.

3.1 Spectral Centrality

The idea of centrality as applied to human communication was first introduced by Bavelas in 1948 [10].
Jarod P.Benowitz defined centrality as a measure in which the nodes of a network are assigned a rank with respect to vertices from most important to least important. The idea of centrality has been applied by the researchers in many areas such as communication network, analysis of the organizational and technological structure.
Complex networks appear frequently in various technological, social and biological scenarios. These networks include the Internet, the World Wide Web, social networks, scientific collaboration networks, lexicon or semantic network, neural networks, food web, metabolic networks , and Proteinprotein interaction networks [8]. In complex networks it is a common phenomenon to focus on the pattern of the interactions between individual components in a system and hence the idea of centrality has been used in large networks.

For example the extension of this measure has been introduced in [5]. In this paper, Crofts and Higman have judged the size of the cluster not by the number of nodes, but by the total weight of connections that they possess [5].

Centrality is an important structural attribute of social network. One common question in network analysis is to determine the most important nodes in the network, also called node or vertex centrality. Due to this, over the years, a great many different measures of centrality have been proposed [2].

There are topological properties from where centrality can be measured. One of the most sophisticated way to measure centrality is by using spectral methods. Spectral methods rely on the eigenvalues of matrix representations of networks, and capture global information on structure. In linear algebra, an eigenvector v of a square matrix A is such that

$$Av = \lambda v$$

where λ is the eigenvalue associated with the eigenvector v.

Networks have distinct spectra (eigenvalue pattern) and hence the spectra are the fingerprints of the network topology [9].

3.2 Eigenvalues and Eigenvectors

Let $A \in C^{nxn}$ be a matrix of complex numbers. A nonzero vector $x \in C^n$ is an eigenvector of Matrix A with corresponding eigenvalue $\lambda \in C$ if x and λ satisfy the equation

$$Ax = \lambda x \tag{3.1}$$

Equation 3.1 has a nonzero solution if and only if the determinant of $A - \lambda I_n$ is zero.

The set of eigenvalues of matrix A can be characterized as roots of the characteristic polynomial $P_A(\lambda) = det(A - \lambda I_n)$

The spectrum of matrix A is defined as the collection of all eigenvalues of A. The degree of

polynomial $P_A(\lambda)$ is n, so, altogether A has n eigenvalues, some of which may be complex, and some eigenvalues may be repeated.

The algebraic multiplicity of an eigenvalue λ of A is the number of times λ is repeated as a root of the characteristic equation $P_A(\lambda) = 0$

The spectrum of the DSM of call graph and the associated eigenvectors can reveal a wealth of structural information about the underlying network as we demostrate in this thesis. The two spectral ranking methods that are used in the thesis are obtained from the eigenvectors associated with selected eigenvalues of the associated DSM.

3.3 Hubs and Authorities

Hubs and Authorities are the two types of important nodes in a network. Hubs are nodes which point to many nodes of the type important. Authorities are these important nodes. From this comes a circular definition: good hubs are those which point to many good authorities and good authorities are those pointed to by many good hubs [2].

3.4 Hypertext Induced Topics Search

Jon Kleinberg, a professor in the Department of Computer Science at Cornell, developed an algorithm called hypertext-induced topic search (HITS), which made use of the link structure of the web in order to discover and rank pages relevant for a particular topic [18]. The HITS ranking relies on an iterative method converging to a stationary solution.
In the network, each node i is assigned two non negative weights. One is *authority weight*(x_i) and the other is *hub weight* (y_i). Initially, each x_i and y_i is given an arbitrary nonnegetive value. Then the weights are updated in the following way:

$$x_i^{(k)} = \sum_{j:(j,i)\in E} y_j^{(k-1)} \quad \text{and} \quad y_i^{(k)} = \sum_{j:(i,j)\in E} x_j^k \quad \text{for } k = 1,2,3\cdots$$

- In the *k*th iteration, node *i* is assigned a new authority weight ($x_i^{(k)}$) equal to the sum of $y_j^{(k-1)}$ where the sum runs over each node *j* which points to node *i*. This is repeated for all nodes in the graph.

- The new hub weight $y_i^{(k)}$ is the sum of ($x_i^{(k)}$), where the sum runs over the nodes *j* to which node *i* points. This is repeated for all nodes in the graph.

- The hub weights are computed from the current authority weights, which in turn were computed from the previous hub weights.

This iteration sequence shows the natural dependency relationship between hubs and authorities. If a node *i* points to many nodes with large x-values, it receives a large y-value. Similarly, if it is pointed to by many nodes with large y-values, it receives a large x-value[20].

After new weights are computed for all nodes, the weights are normalized so that:

$$\sqrt{\sum_{i=1}^{n} x_i^2} = \sqrt{\sum_{i=1}^{n} y_i^2} = 1$$

In order to construct a linear algebra formulation of this method two vectors are created. Vector x^k of the authority weights at iteration k and vector y^k of the hub weights, where

$$\vec{x_k} = [x_k(1), x_k(2), ..., x_k(n)]^T ,$$

$$\vec{y_k} = [y_k(1), y_k(2), ..., y_k(n)]^T .$$

A popular choice of the initialization of the vectors $\vec{x^{(0)}}$ and $\vec{y^{(0)}}$ would be the constant vectors

$$\vec{x^{(0)}} = \vec{y^{(0)}} = [1/\sqrt{n}, 1/\sqrt{n}, ..., 1/\sqrt{n}]^T$$

When A is the adjacency matrix of the directed graph G, the algorithm becomes

$$\vec{x^{(k)}} = c_k A^T \vec{y}_{(k-1)} \ , \qquad \vec{y_k} = c'_k A \vec{x_k} \ ,$$

where c_k and c'_k are the normalization constants chosen such that the sum of the squares of the authority weights, as well as that of the hub weights, is 1 in the kth iteration.
Combining the above two formulas, we get

$$\vec{x^{(k)}} = c_k c'_{(k-1)} A^T A \vec{x}_{(k-1)} \text{ for k} > 1$$
$$\vec{y^{(k)}} = c'_{(k-1)} c_k A A^T \vec{y}_{(k-1)} \text{ for k} > 0$$

Hence, HITS is an iterative power method to compute the dominant eigenvector for AA^T and $A^T A$ [12]. The hub scores are determined by the entries of the dominant eigenvector of AA^T and the authority scores are determined by the entries of the dominant eigenvector of $A^T A$ [2].
We explain the HITS algorithm with an example.
Consider the small directed network in Figure 3.1

The adjacency matrix is given by

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

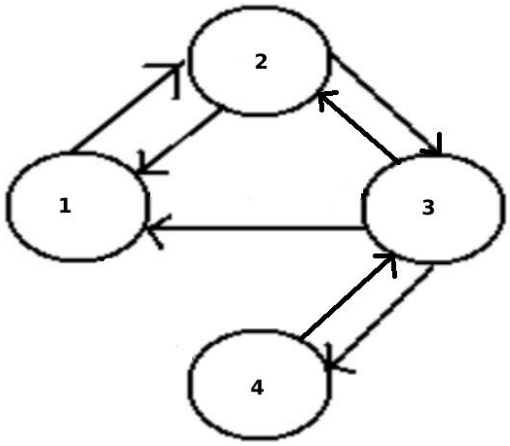


Figure 3.1: Directed network

Just by looking at the graph and the connectivity of the nodes we can see that in terms of hubs there is a tie between the nodes 1,2 and 3. All three nodes are ranked as number one. In terms of authorities, node two is ranked number one followed by node 3. There is a tie between node 1 and node 4. We obtain somewhat different results using the HITS algorithm. The eigenvectors of AA^T and A^TA corresponding to the largest eigenvalue $\lambda_{max} \approx 3.9563$, yield the following ranking for hubs and authorities:

Figure 3.2 shows the ranking for hubs is : 1;3;4;2. The ranking for authority is : 2;3;4;1.

NODE	HUB SCORE(RANK)	AUTHORITY SCORE (RANK)
1	0.168457 (4)	0.65549 (1)
2	0.805799 (1)	0.33507 (4)
3	0.498011 (2)	0.54215 (2)
4	0.27257 (3)	0.40511 (3)

Figure 3.2: Ranking using HITS algorithm

3.4.1 Convergence of the hub-authority computation

In the HITS algorithm there are some numerical issues that need to be pointed out. First, one needs to know if the iterative process converges (to be useful for ranking purposes, the iterations must converge to limiting vectors).

Furthermore, if the process does converge, it is useful to know the conditions required for convergence. For a symmetric matrix $AA^T(A^TA)$, the eigenvalues are real numbers [12]. Since the entries of AA^T and A^TA are nonnegetive, the eigenvalues can be ordered as

$$\lambda_1 = \lambda_2 = \cdots = \lambda_l \geq \lambda_{l+1} \geq \cdots \geq \lambda_n \geq 0$$

The largest eigenvalue (in absolute value) can be of multiplicity l. If $l = 1$, the HITS algorithm converges to unique hub and unique authority vectors.

For $l \geq 2$ the hub and authority vectors need not be unique. The vectors that the HIYS iteration converges to in general will depend on the initial vectors $x^{(0)}$ and $y^{(0)}$.

HUB SCORE

K=1	K=2	K=3	K=4	K=5	K=6	K=7	K=8
0.28571	0.21894	0.19415	0.18241	0.17619	0.17277	0.17086	0.1698
0.71429	0.7663	0.78586	0.79511	0.79994	0.80255	0.80399	0.80479
0.57143	0.54736	0.52699	0.51448	0.50728	0.5032	0.50091	0.49963
0.28571	0.25543	0.25887	0.26426	0.26783	0.26991	0.27108	0.27174

AUTHORITY SCORE

K=1	K=2	K=3	K=4	K=5	K=6	K=7	K=8
0.5547	0.65122	0.66147	0.66034	0.6585	0.65724	0.65648	0.65605
0.5547	0.43414	0.38586	0.36272	0.35042	0.34363	0.33985	0.33774
0.5547	0.5065	0.51448	0.52548	0.53268	0.53685	0.53919	0.5405
0.2773	0.3617	0.38586	0.39528	0.3998	0.40219	0.40349	0.40421

Figure 3.3: Converging to the eigenvector

By looking at Figure 3.2 and Figure 3.3, we see that the hub and authority values are converging to the eigenvector corresponding to the largest eigenvalue.

3.5 Benzi's Method

According to Freeman , the degree of a node is an indicator to represent centrality or importance of a node in a graph. However, the degree of a node represents its connectedness to other nodes in its immediate neighborhood only. Since then several researchers have extended the notion of well-connectedness, i.e many other nodes can be reached from the nodes in question, giving more accurate calculation of node centrality. In particular, researchers have exploited the duality between a graph and its associated matrix.

Given an undirected graph G and its adjacency matrix A which is symmetric, the entries of powers of matrix A contains important topological information about the graph.

The (i,j)th entry of the l_{th} power of the matrix, $A\prime(i,j)$, counts the number of different walks of length l from node i to node j. $A\prime(i,j)$ gives the number of closed walks of the length l that begins and end in node i. We get the identity matrix for $l = 0$. The diagonal can be interpreted as each node in the graph being connected to itself with a walk of length zero. For $l = 1$, a non zero entry in the diagonal indicates a self-loop. The diagonal entries give the degree of nodes for $l = 2$. Thus, the concept of degree of a node can be extended to the concept of well-connectedness or centrality by counting the number of different paths passing through the node. Assuming that the flow of information between the nodes in a graph is more likely to occur along walks of shorter length, the longer walks are penalized by a scaling factor. One of the many scaling factors employed by researchers is the factorial of the length of the walk, giving rise to the infinite series of the matrix exponential [2].

$e^A = 1 + A + A^2/2! + A^3/3! + ... + A'/l! + ...$

The diagonal entries of the matrix e^A provide important information of the node. The larger the value, the more important the node.

Benzi proposed a method where the matrix exponential formulation has been extended to directed graphs. The directed graph is transformed into a symmetric matrix B using the

original matrix and its transpose as shown below.

$$B = \begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix}$$

In the undirected case, i.e the network A, each node has only one role to play in the network. And that is any information that came into the node could leave by the edge.On the other hand, the directed network, B, has two roles for each node: hub and authority. It is unlikely that a node with high hub ranking will also have high authority ranking, but each node can still be seen as acting in both of these roles. In the network B, the two aspects of each node are separated. Nodes 1 to n in $V(B)$ represent the original nodes in their role as hubs, and nodes $n+1$ to $2n$ in $V(B)$ represent the original nodes in their role as authorities [2].

The eigenvalues of B can be ordered as $\lambda_1 > \lambda_2 \geq \lambda_3 \geq ... \geq \lambda_{2n}$

Since B is symmetric matrix its eigenvectors span the $\Re^{2n}$. Therefore, it has orthogonal eigenvectors and the matrix B can be written as:

$B = \sum_{i=1}^{2n} \lambda_i u_i u_i^T$

where $u_1, u_2, ..., u_{2n}$are the normalized eigenvectors of B. Taking the exponential of B, we get:

$e^B = \sum_{i=1}^{2n} e_i^{\lambda} u_i u_i^T,$

The diagonal entries of e^B indicates the hub and authority rankings.

The use of the matrix exponential for ranking hubs and authority amounts to using the entries of all eigenvectors of B, weighted by the exponential of the corresponding eigenvalues [2].

$e^B = \sum_{i=1}^{k} e_i^{\lambda} u_i u_i^T,$

where 1 < k < n

Chapter 4

Numerical Experiments

4.1 Measuring Centrality

As discussed in Chapter 3, we are trying to determine the centrality of the components of scientific software. Instead of using degree-based centrality we are using spectral method for the analysis. Spectral methods rely on the eigenvalues of matrix representations of networks, and capture global information on structure. The basic premise is that networks have distinct spectra and hence the spectra are 'fingerprints' of network topology [9]. It is expected that spectral method will provide more accurate results.

The matrix exponential method for computing hubs and authorities is compared to HITS algorithm both on small examples and on three open source scientific research software. In this Chapter we will show the results of our experiments that we have computed and we will then discuss and compare the pattern of those results.

All the numerical experiments presented in this Chapter are performed in octave on a two core Intel Xeon CPU running at 2.4 GHz.

For eigenvalue decomposition we used octave function *eig*. The matrices are small enough so the running time was not recorded.

4.2 Experimental Results

4.2.1 Small examples

In this section we compare degree-based, HITS and our proposed method to obtain hub and authority rankings. The following two examples are taken from the paper "Linear Algebra and its Application" by Michele Benzi, Ernesto Estrada and Christin Klymko.

Example # 1:

$$A = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

The corresponding bipartite graph is shown in Figure 4.1.

A plot of the eigenvalues of the matrix A can be found in Figure 4.2

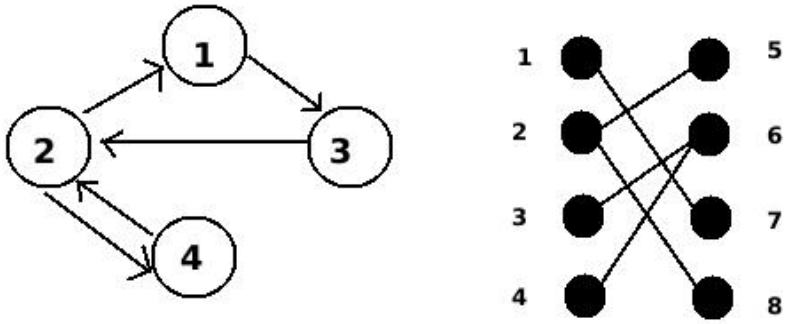


Figure 4.1: Example 1 - bipartite graph

Table 4.1: Degree centrality score

Node	Outdegree	Indegree
1	1	1
2	2	2
3	1	1
4	1	1

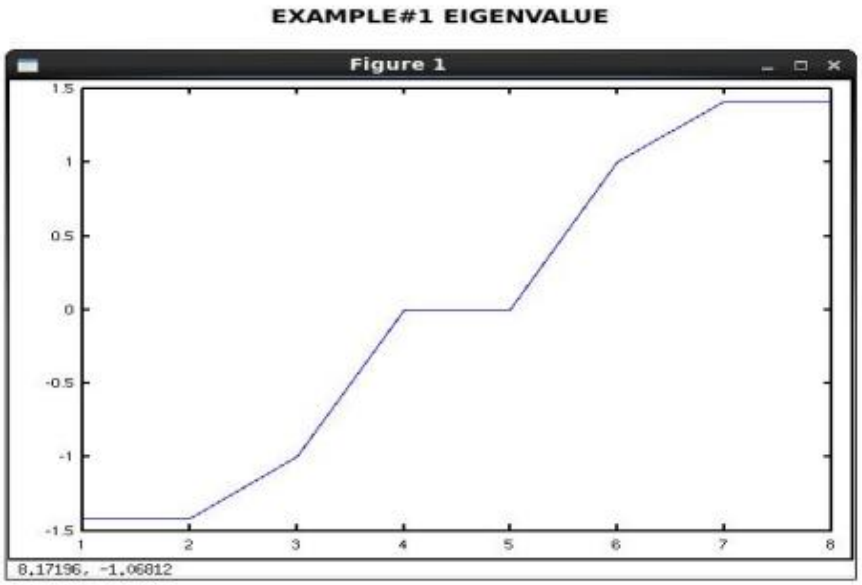


Figure 4.2: Example 1 - graph plot of eigenvalues

EXAMPLE # 1

Change in ranking using Benzi's Method:

0	0	1.35914	1.35914	1.35914	1.54308	1.54308	1.54308
0	2.05663	2.05663	2.05663	2.05663	2.05663	2.05663	2.17818
1.02831	1.02831	1.02831	1.52831	1.52831	1.52831	1.52831	1.58909
1.02831	1.02831	1.02831	1.52831	1.52831	1.52831	1.52831	1.58909
0	1.02831	1.02831	1.02831	1.52831	1.52831	1.52831	1.58909
2.05664	2.05663	2.05663	2.05663	2.05663	2.05663	2.05663	2.17818
0	0	1.35914	1.35914	1.35914	1.54308	1.54308	1.54308
0	1.02831	1.02831	1.02831	1.52831	1.52831	1.52831	1.58909

Figure 4.3: Example 1 - change in ranking using benzi's method

The hubs and authorities that are determined by using in-degree and out-degree are displayed in Table 4.1. Under this criterion, the hub and authority rankings are both 2;1,3,4(tie). While it is intuitive that node 2 should be given a high score for both hub and authority, just by looking at the degrees does not allow one to distinguish the remaining nodes.

According to HITS algorithm, the largest eigen value of AA^T and A^TA is 2. The rank-

Table 4.2: Ranking using HITS algorithm

Node	Hub Rank	Authority Rank
1	0	0.333
2	0.5	0.333
3	0.25	0
4	0.25	0.333

ing suggested by the dominant eigen vectors is reported in Table 4.2. In terms of hubs, node 2 is ranked number one followed by node 3 and 4 which are ranked number two. On the other hand, in terms of authorities, node 1,2 and 3 are all ranked number 1.

In Figure 4.3 we report the change in ranking of the hubs and authority scores obtained by the method of Benzi et al. The first four values represent the hub scores and the bottom four represents the authority scores. By looking at the first column, we can see that node 3 and 4 are ranked number 1 followed by node 1 and 2. On the other hand, node 2 of column 2 is ranked number one. From column three onwards node 2 is ranked number 1. Ranking of the last two columns are the same. Node 3 and 4 are ranked number two followed by node one.

In terms of authority,the ranking is as following. Node 2 is ranked number 1 followed by node 1 and 4.

With Benzi's method, the hub ranking of the nodes is the same as in HITS. The hub ranking is given as, rank one: node 2, rank two: node 3 and node 4, rank three: node 1. However, in authority ranking the results are much more transparent. Node 2 is the clear winner than being part of a three-way tie for first place:{2;1,4(tie);3}.

The method based on the matrix exponential is able to identify a top authority node by making use of additional spectral information.

Example # 2:

$$A = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 \end{pmatrix}$$

The corresponding bipartite graph is shown below.

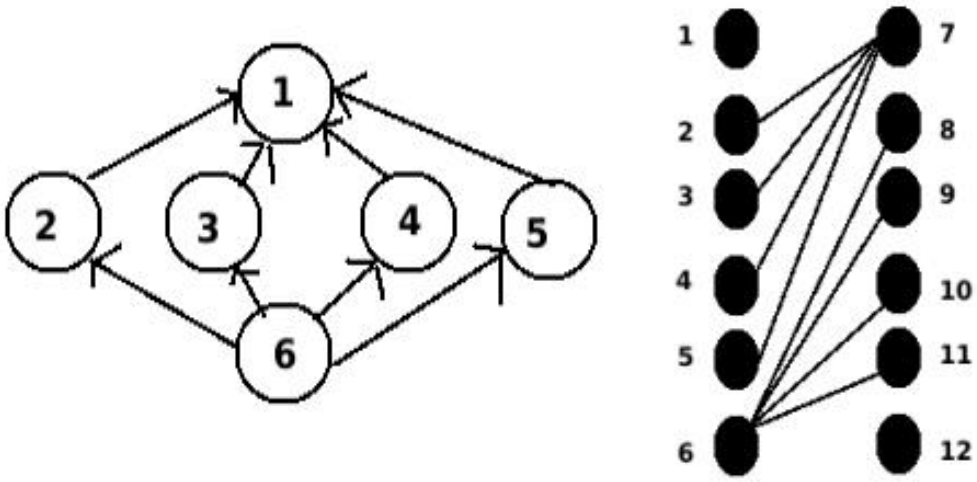


Figure 4.4: Example 2 - bipartite graph

If hubs and authorities are determined using only in-degrees and out-degrees, the result is:

A plot of the eigenvalues of the matrix A can be found in Figure 4.5.

Table 4.3: Degree centrality score

Node	Outdegree	Indegree
1	0	4
2	1	1
3	1	1
4	1	1
5	1	1
6	4	0

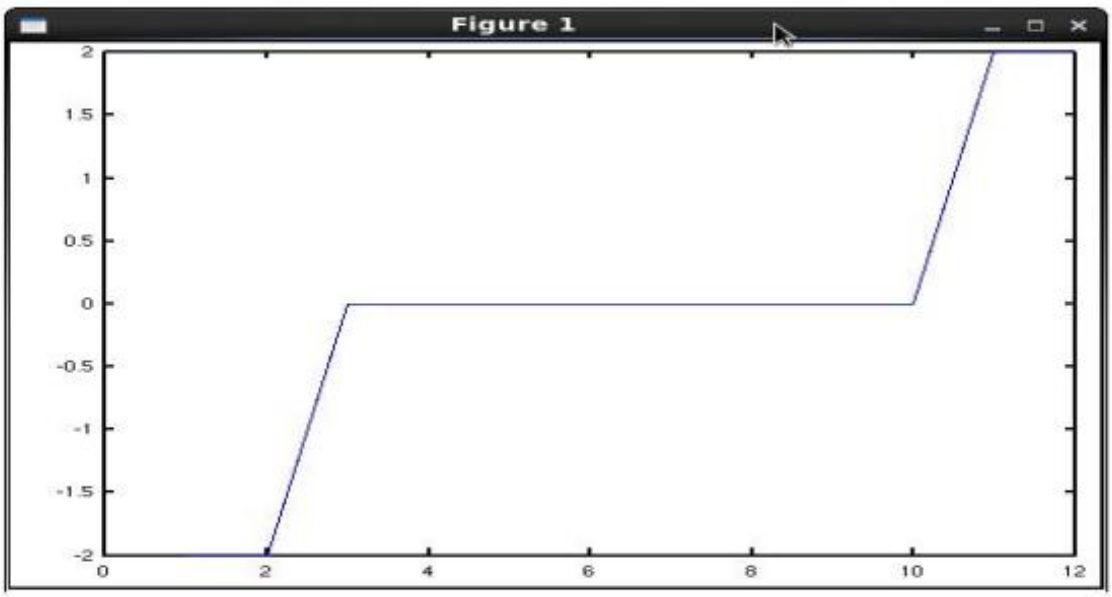


Figure 4.5: Example 2 - graph plot of eigenvalues

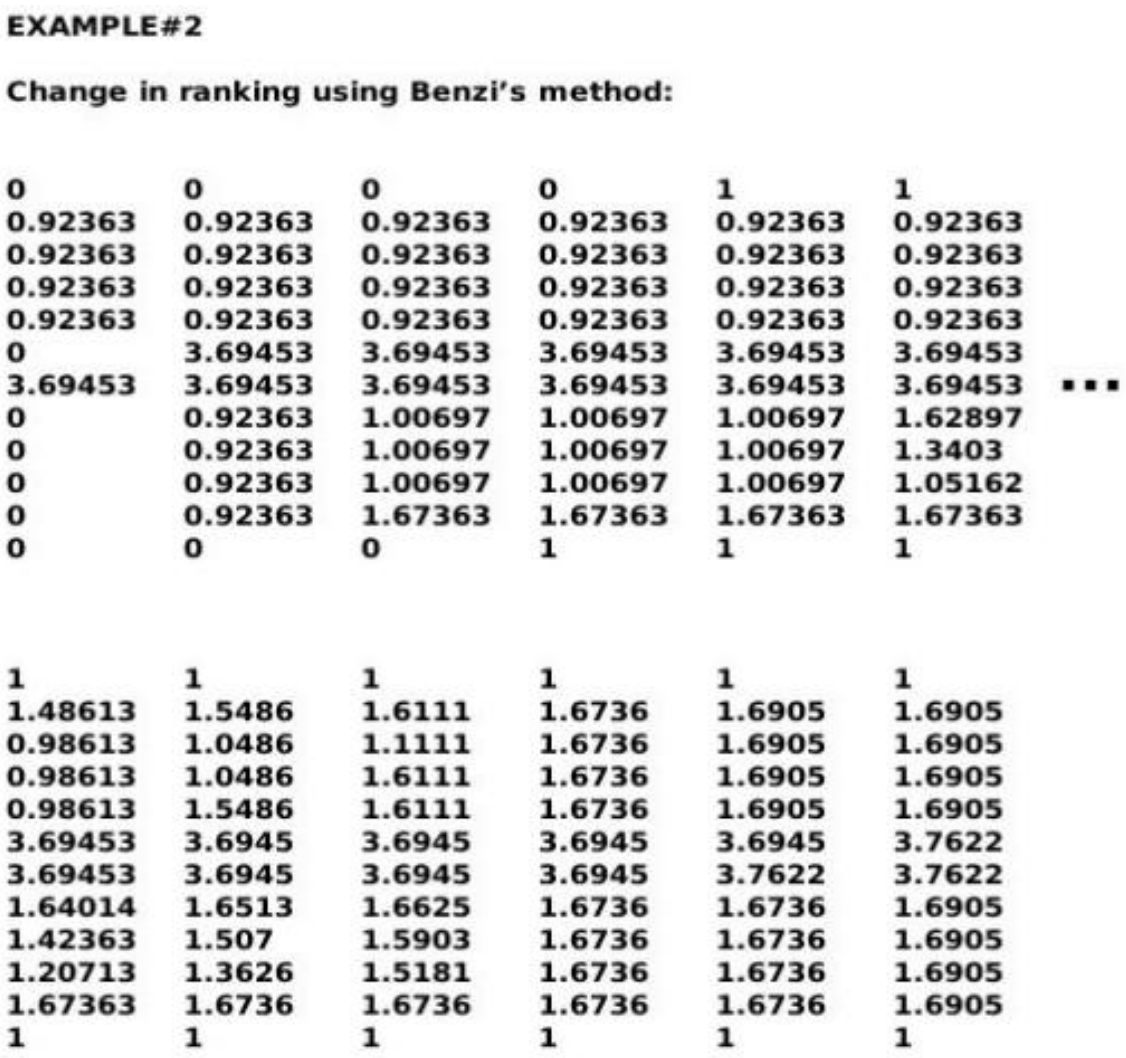


Figure 4.6: Example 2 - change in ranking using benzi's method

By looking at Table 4.4 and figure 4.6 we see that the hub ranking provided by HITS algorithm and and Benzi's algorithm are same. In case of authority, HITS does not differentiate between node 1 and nodes 2,3,4 and 5. When e^B is used to calculate the hub and authority scores, node 1 gets a higher authority ranking than all the other nodes. This appears as a failure of HITS.

Table 4.4: Ranking using HITS algorithm

Node	Hub rank	Authority rank
1	0	.2
2	.125	.2
3	.125	.2
4	.125	.2
5	.125	.2
6	.5	0

Example # 3:

This example has been taken from the paper ” On Ranking Components in Scientific Software”.

Let the adjacency matrix be:

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

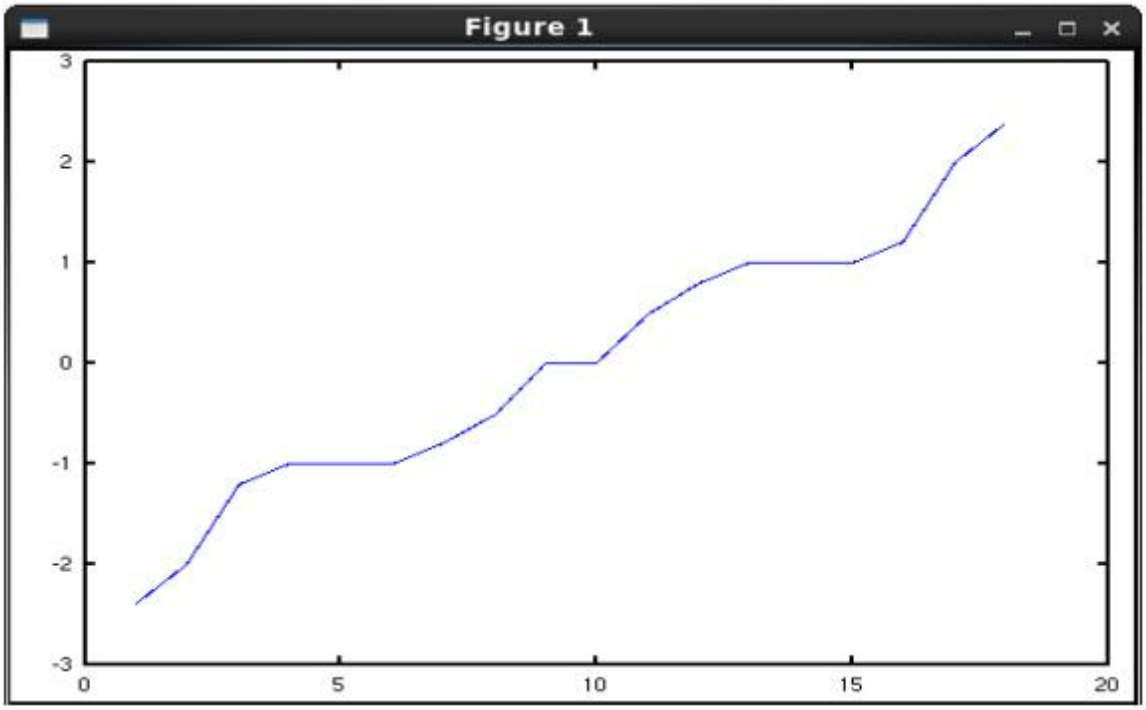


Figure 4.7: Example 3 - graph plot of eigenvalues

Figure 4.8 portrays the directed network of the above adjacency matrix. Based on the out-degree, nodes 4 and 6 are tied for rank one. All the other nodes have out-degree one and therefore tied for rank two. In terms of in-degree, node 6 is ranked number one followed by nodes 1,4,5 and 9. The remaining nodes, with degree 1 are tied for rank three.

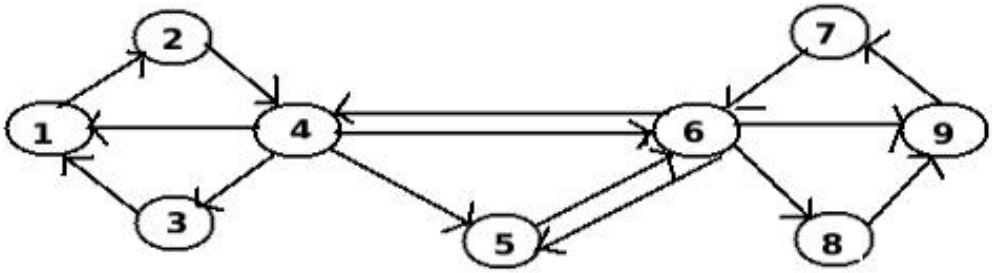


Figure 4.8: Example 3 - directed graph

The largest eigenvalue of matrix AA^T and A^TA is 5.6. The ranking suggested by the dominant eigenvectors is shown in Figure 4.9. In terms of hubs, node 4 is ranked number one followed by node 6 which is ranked number two. As in the out-degree based hub rank-

Hub and authority ranking using HITS algorithm

NODE	HUB RANK	AUTHORITY RANK
1	0	0.36464
2	0.13015	0
3	0.15387	0.29971
4	0.71022	0.30841
5	0.19643	0.5532
6	0.60071	0.46549
7	0.19643	0
8	0.13015	0.25349
9	0	0.30841

Figure 4.9: Example 3 - ranking using HITS algorithm

EXAMPLE#3

Change in ranking using Benzi's method

7.43E-038	4.50E-036	7.52E-032	0.049978	0.076514	1.35914	1.35914	1.3591	1.3591
9.06E-002	2.93E-001	3.04E-001	0.304247	0.970134	0.98381	1.16719	1.3841	1.3841
1.27E-001	2.30E-001	6.84E-001	0.683554	0.683554	0.68355	1.21258	1.3832	1.3832
2.70E+000	3.60E+000	3.73E+000	3.729238	3.729238	3.72924	3.81087	3.8966	3.8966
2.06E-001	4.42E-001	1.00E+000	0.999541	0.999541	0.99954	1.04162	1.071	1.1083
1.93E+000	3.70E+000	3.70E+000	3.70433	3.70433	3.70433	3.73263	3.8416	3.8416
2.06E-001	4.42E-001	1.00E+000	0.999541	0.999541	0.99954	1.04162	1.071	1.1083
9.06E-002	2.93E-001	3.04E-001	0.304247	0.970134	0.98381	1.16719	1.3841	1.3841
5.75E-033	6.01E-033	1.27E-030	1.309154	1.309985	1.35914	1.35914	1.3591	1.3591
7.11E-001	1.12E+000	1.81E+000	1.812615	1.812615	1.81262	2.13383	2.1835	2.1835
5.57E-038	5.02E-036	1.08E-031	0.049978	0.076514	1.35914	1.35914	1.3591	1.3591
4.80E-001	7.09E-001	7.91E-001	0.791401	0.791401	0.7914	0.92585	1.2201	1.5286
5.09E-001	1.31E+000	1.33E+000	1.326805	1.992692	2.00637	2.11771	2.1809	2.1809
1.64E+000	1.67E+000	1.73E+000	1.730731	1.730731	1.73073	2.07011	2.0749	2.3833
1.16E+000	2.09E+000	2.94E+000	2.943308	2.943308	2.94331	2.96886	2.9774	2.9774
1.35E-031	1.70E-031	1.14E-030	1.309154	1.309985	1.35914	1.35914	1.3591	1.3591
3.44E-001	7.91E-001	7.93E-001	0.793033	0.793033	0.79303	0.83964	1.2137	1.5222
5.09E-001	1.31E+000	1.33E+000	1.326805	1.992692	2.00637	2.11771	2.1809	2.1809

1.3591	1.3591	1.3591	1.3591	1.4678	1.5431	1.5431	1.5431	1.5431
1.3841	1.4578	1.4964	1.5883	1.5883	1.5883	1.5893	1.5931	1.5939
1.3832	1.4412	1.5525	1.5525	1.5525	1.5525	1.5909	1.5928	1.5939
3.8966	3.9257	3.9429	3.9429	3.9429	3.9429	3.9535	3.9704	3.994
1.571	1.581	1.5898	1.5898	1.5898	1.5898	1.6369	1.6413	1.6431
3.8416	3.8786	3.8846	3.8846	3.8846	3.8846	3.8848	3.9179	3.9348
1.571	1.581	1.5898	1.5898	1.5898	1.5898	1.6369	1.6413	1.6431
1.3841	1.4578	1.4964	1.5883	1.5883	1.5883	1.5893	1.5931	1.5939
1.3591	1.3591	1.3591	1.3591	1.4344	1.5431	1.5431	1.5431	1.5431
2.1835	2.2004	2.268	2.268	2.268	2.268	2.3265	2.3342	2.3404
1.3591	1.3591	1.3591	1.3591	1.4678	1.5431	1.5431	1.5431	1.5431
1.5535	1.6535	1.6818	1.6818	1.6818	1.6818	1.6887	1.693	1.6972
2.1809	2.2023	2.2258	2.3177	2.3177	2.3177	2.3192	2.3342	2.3386
2.4082	2.4098	2.4813	2.4813	2.4813	2.4813	2.4862	2.4869	2.5012
2.9774	2.9803	2.9857	2.9857	2.9857	2.9857	3.0576	3.075	3.0852
1.3591	1.3591	1.3591	1.3591	1.4344	1.5431	1.5431	1.5431	1.5431
1.5471	1.6742	1.684	1.684	1.684	1.684	1.6842	1.6925	1.6955
2.1809	2.2023	2.2258	2.3177	2.3177	2.3177	2.3192	2.3342	2.3386

Figure 4.10: Example 3 - change in ranking using benzi's method

ing, nodes 5 and 7 are ranked number three. In case of authority scores, we can see that node 5 is ranked number one followed by node 6. This ranking is more consistent with the

actual topology of the graph as node 5 is being pointed by important hub node 6 receives the highest authority ranking. Nodes 1 and 3 are ranked number 3.

In Figure 4.10 we report the change in hub and authority scores using Benzi's method. The ranking obtained by Benzi et at. gives a similar picture as in the HITS ranking. In terms of hubs, node 4 has rank number one followed by node 6. Nodes 5 and 7 are ranked as three. Rank four is given to nodes 2,3 and 8 followed by nodes 1 and 9. The authority ranking is given as, rank one:node 6, rank two:node 5, rank three: nodes 1,4,9, rank four: nodes 3 and 8, rank five: nodes 2,7.
Comparing HITS with Benzi's method, we see that Benzi's method gives node 5 second highest authority score which is one rank lower than the one provided by HITS.

4.2.2 Large examples

Computational infrastructure for Operation Research (COIN-OR) is one of the largest and most widely studied open source communities for scientific research software. We studied open source software projects from COIN-OR.
The three scientific computing C/C++ projects that we have used for our experiments are ADOL-C, CppAD and CSparse. In this section we will first briefly introduce these softwares one by one and then display the results that we got from our experiments and discuss and compare those results. In order to work with these softwares we first downloaded the original source codes of these scientific computing projects. Later we used the tool *Understand* for data extraction.
The DSM of the call graphs provides a convenient tool so that linear algebraic techniques can be applied to identify important callers and callees through the calculation of matrix exponentials. We also tried to explore the extension of HITS and Benzi et al. methods to incorporate edge weights in determining the relative ranking of the function nodes. For instance, if a function calls another function multiple times from within its body, it is im-

portant to incorporate this information in the analysis as edge weights.

ADOL-C:

Automatic Differentiation by OverLoading in C++ (ADOL-C) is an open-source package for the automatic differentiation of C and C++ programs [14]. The first and higher derivatives of vector functions are evaluated using operator overloading method by ADOL-C [27] It is developed by a team of researchers from Argonne National Lab,Dresden University of Technology, and Humboldt University over a period of more than 20 years.

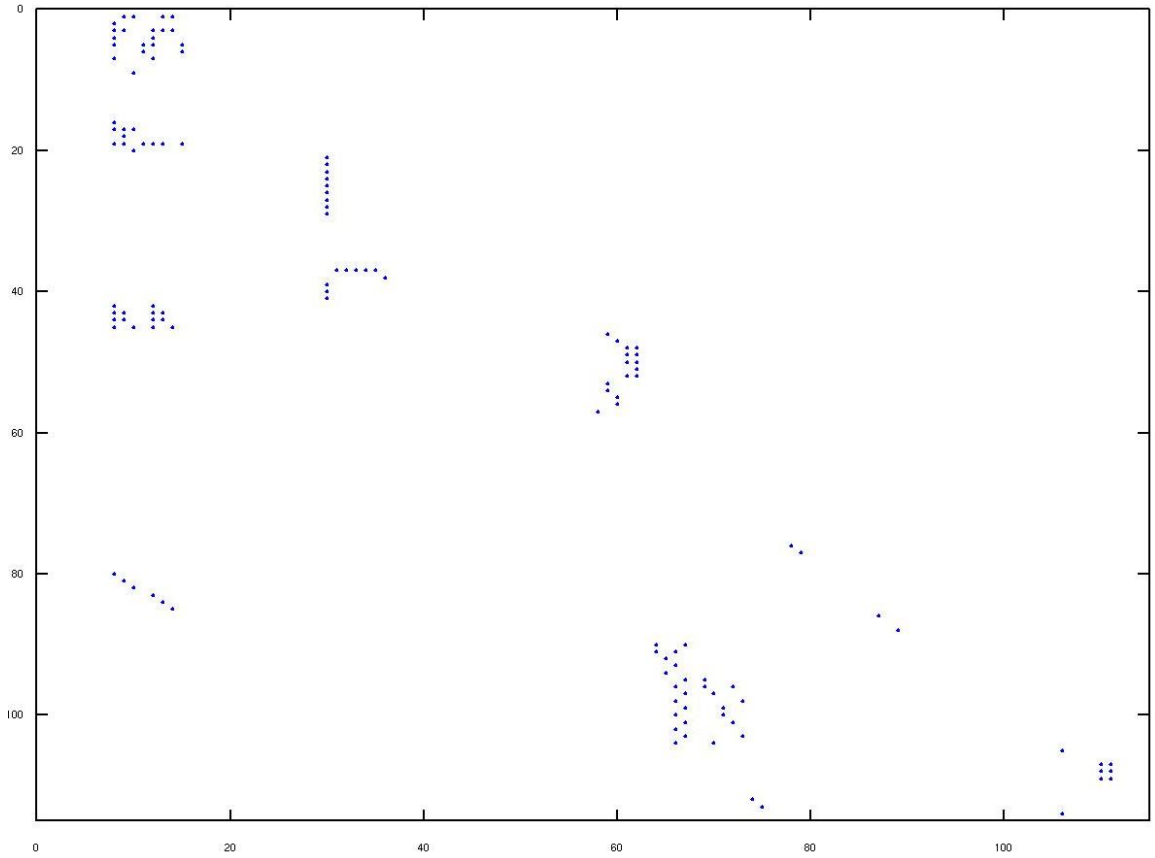


Figure 4.11: ADOL-C, sparse matrix

Figure 4.11 shows the sparse matrix of ADOL-C. A plot of the eigenvalues of the bipartite matrix of ADOL-C can be found in Figure 4.12. Figure 4.13 portrays the architectural dependency of ADOL-C. The top 5 hubs and authorities of ADOL-C, as determined using the diagonal entries of e^B and HITS are shown in Figure 4.14 and Figure 4.15. The algo-

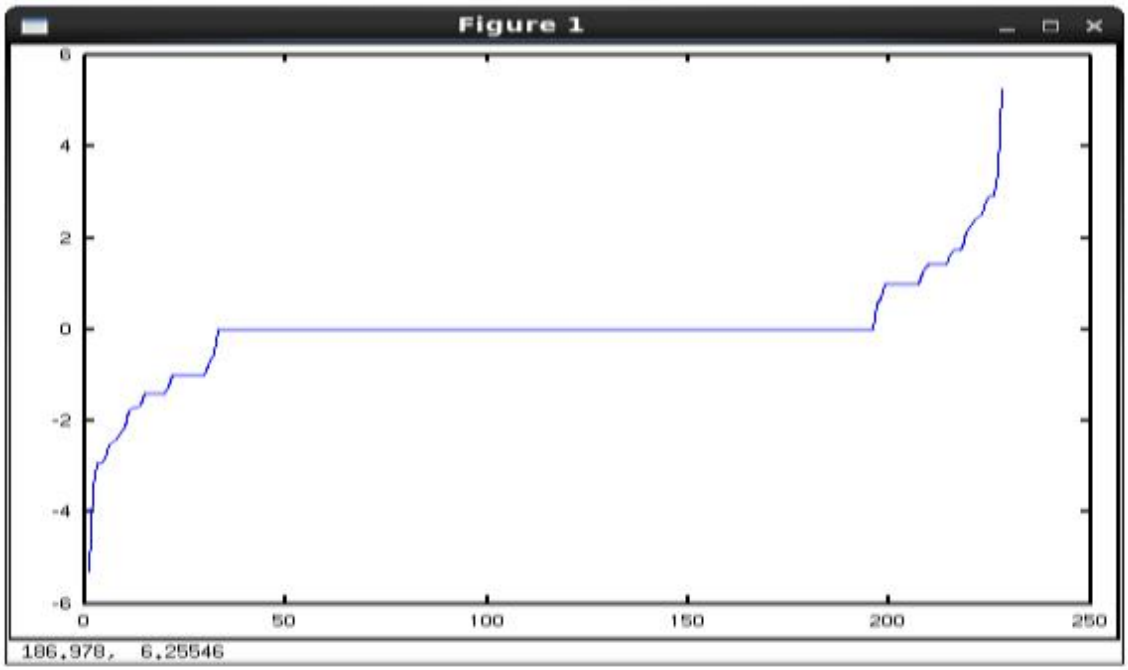


Figure 4.12: ADOL-C, graph plot of eigenvalues

rithms have been applied on both non-weighted call graph as well as weighted call graph. We observe that there is a good deal of agreement between the e^B rankings and the HITS rankings. In fact, both methods ranked the functions myalloc1(), myfree1(), myalloc2(), myfree2(), myfree3() as number one, two , three, four and five respectively in terms of hubs.

Concerning the authorities, just like hubs, we get similar picture for both the methods.

Comparing Benzi's method for non-weighted call graph with weighted call graph, in Figure 4.15, we can see that for the non-weighted call graph, in terms of authority score, the two functions free_loc() and next_loc() both are ranked number three. On the other hand, for weighted call graph, the rankings are much more transparent and clearly shows that the function free_loc() is ranked number three and the function next_loc() is ranked as number two.

In this example, the method based on the matrix exponential, applied on weighted call graph, has produced more transparent and distinct authority score, relative to rankings ob-

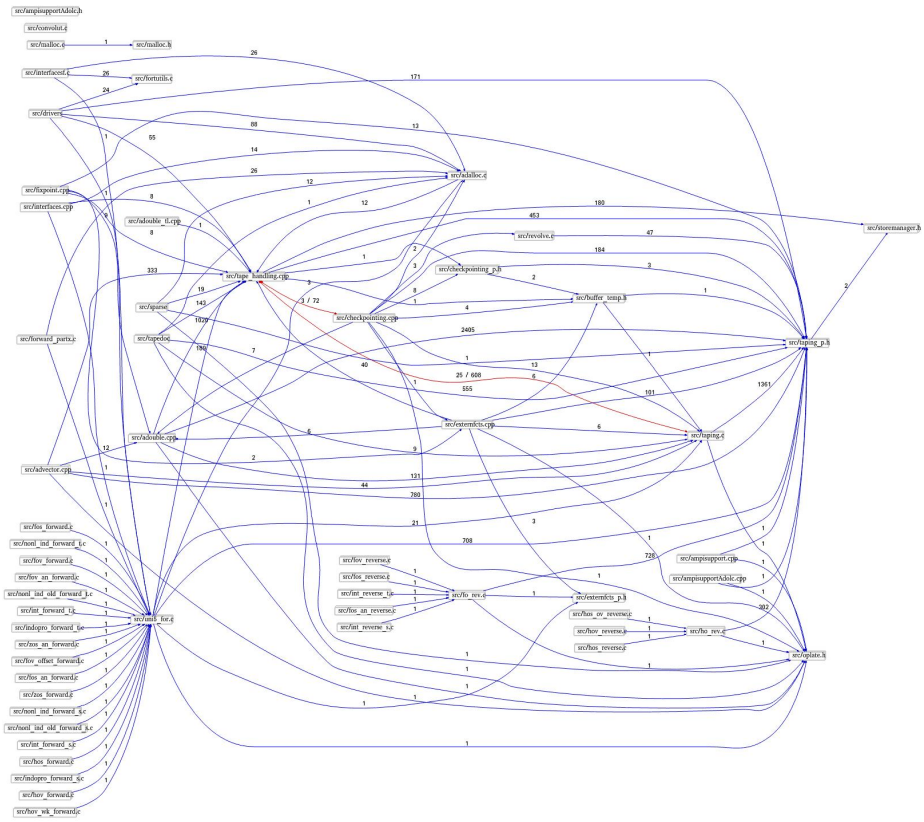


Figure 4.13: ADOL-C, dependency graph

Ranking using HITS Algorithm -ADOL-C

Function Name	Hub rank	Indegree	Outdegree
myalloc1()	0.61407	0	13
myfree1()	0.52585	0	10
myalloc2()	0.38774	0	7
myfree2()	0.33073	0	6
myfree3()	0.1675	0	4

Function Name	Authority rank	Indegree	Outdegree
ensure_block()	0.40339	2	0
consolidateBlocks()	0.38334	2	0
free_loc()	0.35165	2	0
next_loc()	1.35165	2	0
grow()	0.27496	1	0

Ranking using HITS Algorithm (WEIGHTED)-ADOL-C

Function Name	Hub rank	Indegree	Outdegree
size()	0.89065	0	5
maxSize()	0.45469	0	4

Function Name	Authority rank	Indegree	Outdegree
ensure_block()	0.72037	2	0
consolidateBlocks()	0.35911	2	0
free_loc()	0.35911	2	0
next_loc()	0.35911	2	0
grow()	0.30689	1	0

Figure 4.14: ADOL-C, ranking using HITS algorithm

tained from non-weighted call graph, by making use of the additional information.

CSPARSE

CSparse is a C library which implements a number of direct methods for sparse linear systems, by Timothy Davis ($https://people.sc.fsu.edu/\ jburkardt/c_src/csparse/csparse.html$). It is a library of functions, with regard to our call graph we can extract valuable information

Top 5 values

HUB SCORE	FUNCTION	INDEGREE	OUTDEGREE
110.6378	myalloc1()	0	13
81.1323	myfree1()	0	10
44.1101	myalloc2()	1	7
32.0924	myfree2()	0	6
8.232	myfree3()	0	4

Top 5 values (WEIGHTED)

HUB SCORE	FUNCTION NAME	INDEGREE	OUTDEGREE
5079.78	setStoreManagerControl()	0	1
1323	setStoreManagerControl()	1	0

Bottom 5 Values

HUB SCORE	FUNCTION	INDEGREE	OUTDEGREE
3.39E-180	operator>=()	2	0
3.39E-180	operator>=()	2	0
2.77E-172	operator>()	1	0
5.62E-141	operator>()	2	0
9.97E-64	)OLC_get_sparse_jabobian	1	0

Bottom 5 values (WEIGHTED)

HUB SCORE	FUNCTION NAME	INDEGREE	OUTDEGREE
-2.89E-117	operator>()	2	0
-1.66E-054	operator>=()	0	2

Top 5 values

AUTHORITY	FUNCTION	INDEGREE	OUTDEGREE
47.7438	ensure_block()	2	0
43.116	consolidateBlocks()	2	0
36.281	free_loc()	2	0
36.281	next_loc()	2	0
22.1828	grow()	1	0

Top 5 values (WEIGHTED)

AUTHORITY	FUNCTION NAME	INDEGREE	OUTDEGREE
3323.13	ensure_block()	2	0
827.82	next_loc()	2	0
826.82	free_loc()	2	0
825.82	consolidateBlocks()	2	0
603.1	grow()	1	0

Bottom 5 Values

AUTHORITY	FUNCTION	INDEGREE	OUTDEGREE
9.23E-315	myfree1()	0	10
1.18E-283	myfree2()	0	6
1.20E-253	myfree3()	0	4
1.03E-221	myfreel2()	0	3
6.98E-192	GauszSolve()	0	4

Bottom 5 values (WEIGHTED)

AUTHORITY	FUNCTION NAME	INDEGREE	OUTDEGREE
7.35E-286	myfree1()	0	10
3.84E-256	myfree2()	0	6
7.64E-223	myfree3()	0	4
2.70E-190	myfreel2()	0	3
9.02E-062	filewrite_start()	1	0

Figure 4.15: ADOL-C, ranking using benzi's method

about the importance of the functions implemented in the library using our technique. The sparse matrix of CSparse can be found in Figure 4.16. Figure 4.17 portrays a plot of the eigenvalues of the bipartite matrix of csparse.

The architectural dependency graph of CSparse can be found in Figure 4.18. Figure 4.19 and 4.20 shows the highest hub and authority scores for the functions in CSparse library computed using HITS and Benzi's method for both non-weighted and weighted call graphs. We see less agreement between HITS and Benzi's method. When comparing the two methods for the non-weighted call graph, the results are as follows:

In terms of hubs, both methods agree that the function cs_malloc() is the most important hub and ranked it as number one. However, the method based on e^B identifies the functions cs_spalloc() and cs_free() as third most important hub and fourth most important hub respectively. Whereas, using HITS algorithm, the functions cs_free() and the function cs_spalloc() are ranked as number three and four. The two methods agree on the hub and ranked the function cs_spfree() as number five. Concerning authorities, both the methods are giving similar picture.

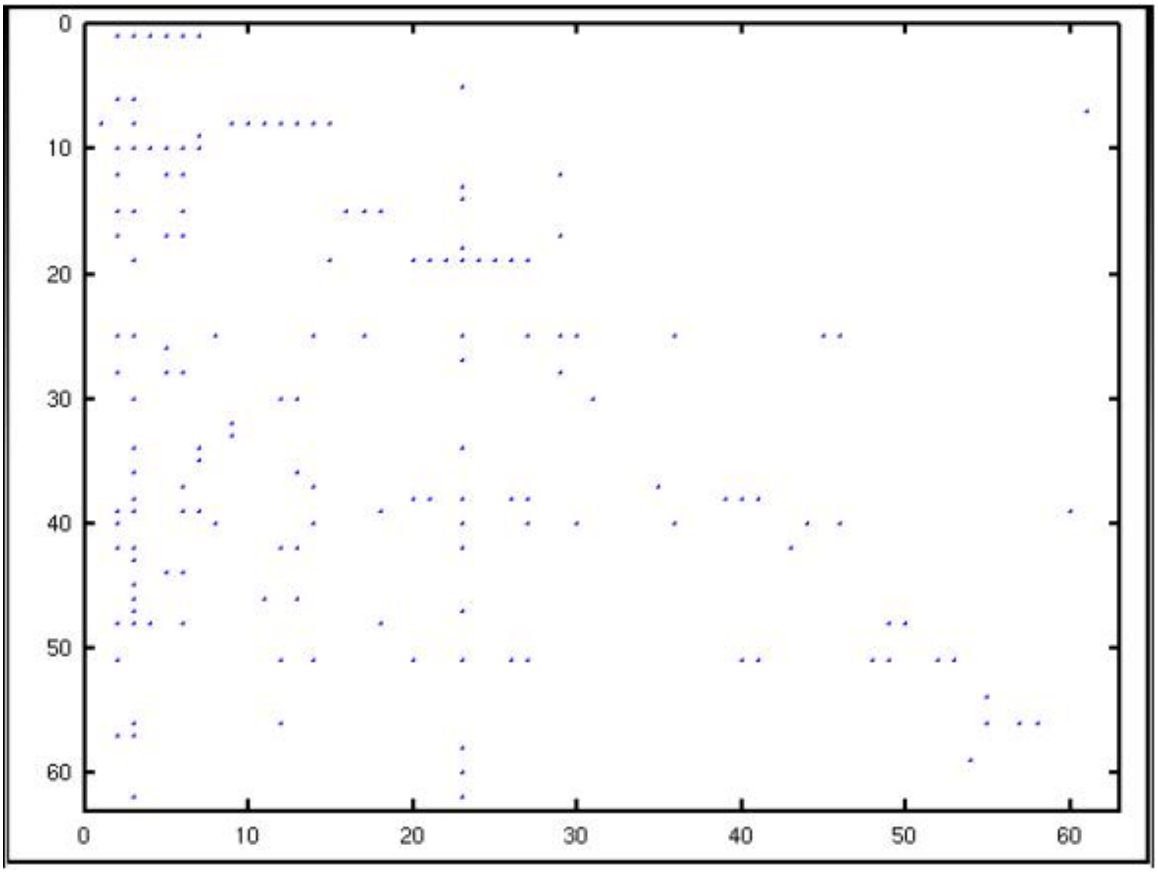


Figure 4.16: CSparse, sparse matrix

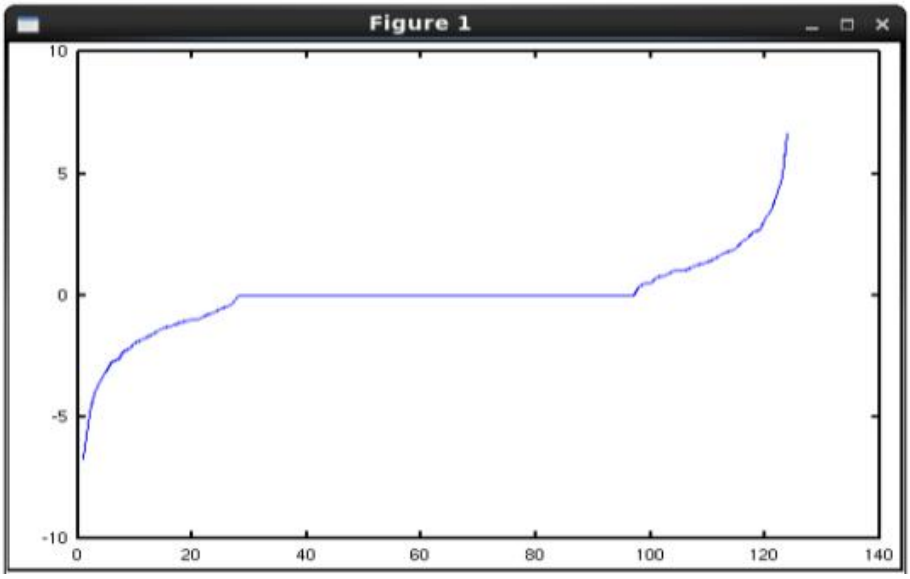


Figure 4.17: CSparse, graph plot of eigenvalues

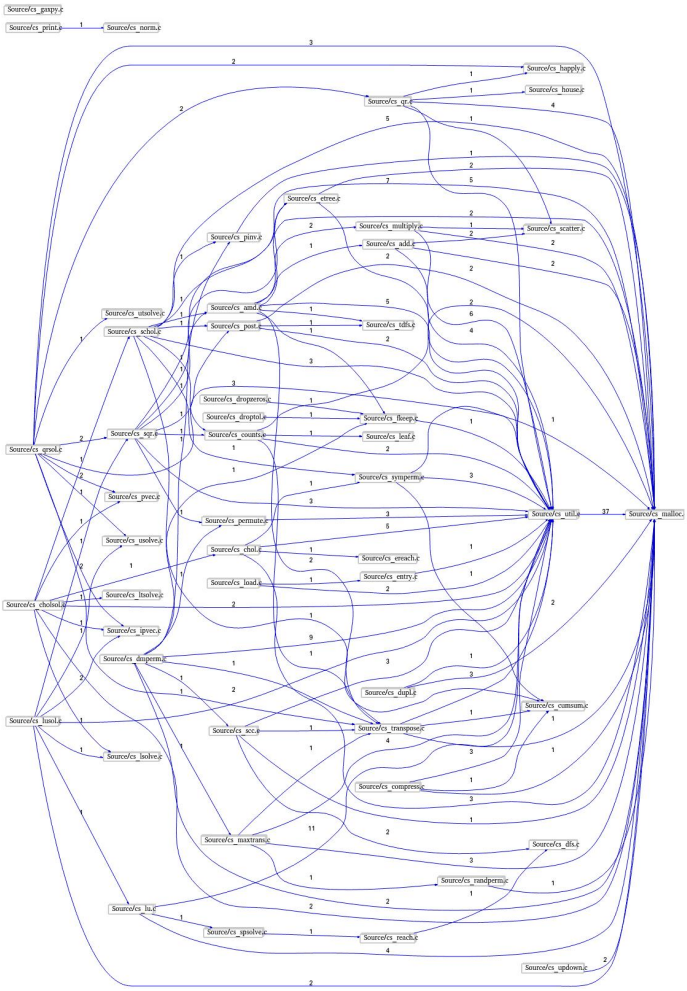


Figure 4.18: CSparse, dependency graph

Ranking using HITS Algorithm – Csparse

Function Name	*Hub rank*	*Indegree*	*Outdegree*
cs_malloc()	0.56084	0	20
cs_calloc()	0.45185	0	14
cs_free()	0.35328	0	15
cs_spalloc()	0.26146	2	10
cs_spfree()	0.19786	1	4

Function Name	*Authority rank*	*Indegree*	*Outdegree*
cs_schol()	0.34593	12	0
cs_qrsol()	0.28098	13	0
cs_add()	0.24935	6	1
cs_multiply()	0.24935	6	1
cs_maxtrans()	0.24622	6	2

Ranking usinh HITS Algorithm (WEIGHTED) – Csparse

Function Name	Hub rank	Indegree	Outdegree
cs_free()	0.61624	0	15
cs_malloc()	0.57886	0	20
cs_ndone()	0.30922	0	2
cs_calloc()	0.22041	0	14
cs_sprealloc()	0.17243	1	4

Function Name	Authority rank	Indegree	Outdegree
cs_lu()	0.38142	6	1
cs_sfree()	0.30911	1	5
cs_qr()	0.27996	7	1
cs_schol()	0.26909	12	0
cs_spsolve()	0.2576	1	1

Figure 4.19: CSparse, ranking using HITS algorithm

Comparing the results of the non-weighted and weighted call graphs (Benzi's method), we see that, in terms of hubs, the function cs_malloc() is ranked number one for the non-

Ranking using Benzi's Method - CSparse

Top 5 values

Hub rank	FUNCTION	INDEGREE	OUTDEGREE
196.6689	cs_malloc()	0	20
127.6602	cs_calloc()	0	14
42.7449	cs_spalloc()	2	10
24.4783	cs_sfree()	1	5
16.5754	cs_spfree()	1	4

Bottom 5 Values

Hub rank	FUNCTION	INDEGREE	OUTDEGREE
9.24E-32	cs_dupl()	3	0
3.33E-32	cs_qrsol()	13	0
1.30E-32	cs_droptol()	1	0
7.15E-31	cs_maxtrans()	6	0
4.80E-31	cs_dropzeros()	1	0

Top 5 values

Authority rank	FUNCTION	INDEGREE	OUTDEGREE
74.82437	cs_schol()	12	0
49.3627	cs_qrsol()	13	0
38.875	cs_add()	6	1
38.875	cs_multiply()	6	1
37.9059	cs_maxtrans()	6	2

Bottom 5 Values

Authority rank	FUNCTION	INDEGREE	OUTDEGREE
4.57E-61	cs_tdfs()	0	2
1.30E-36	cs_pvec()	0	0
6.16E-33	cs_utsolve()	0	1
1.85E-33	cs_dfs()	0	2
7.24E-32	cs_usolve()	0	2

Ranking using Benzi's Method (WEIGHTED) - CSparse

Top 5 values

Hub rank	FUNCTION NAME	INDEGREE	OUTDEGREE
37885	cs_free()	0	15
33428	cs_malloc()	0	20
9539	cs_ndone()	0	2
4849	cs_calloc()	0	14
2966	cs_sprealloc()	1	4

Bottom 5 Values

Hub rank	FUNCTION NAME	INDEGREE	OUTDEGREE
7.28E-030	cs_load()	9	
7.20E-030	cs_lusol()	9	0
6.55E-030	cs_droptol()	1	0
7.41E-029	cs_maxtrans()	6	0
2.41E-029	cs_dropzeroes()	1	0

Top 5 values

Authority rank	FUNCTION NAME	INDEGREE	OUTDEGREE
14513	cs_lu()	6	1
9532.615	cs_sfree()	1	5
7819.13	cs_qr()	7	1
7224.061	cs_schol()	12	0
6619.87	cs_spsolve()	1	1

Bottom 5 Values

Authority rank	FUNCTION NAME	INDEGREE	OUTDEGREE
3.14E-057	cs_tdfs()	0	2
5.68E-032	cs_house()	0	1
4.73E-031	cs_dfs()	0	2
1.82E-031	cs_usolv()	0	2
1.13E-031	cs_pvec()	0	0

Figure 4.20: CSparse, ranking using benzi's method

weighted call graph whereas the function cs_free() is ranked as number one in weighted call graph.

In terms of authority, two functions in the non-weighted call graph has a tie. Both the functions cs_add() and cs_multiply() are ranked as number three. But for the weighted call graph, we can see that all the five functions has distinct ranks making it unambiguous.

CppAD

Given a C++ algorithm that computes function values, CppAD generates an algorithm that computes its derivative values. A brief introduction to Algorithmic Differentiation(AD) can be found in Wikipedia . The web site "autodiff.org" is dedicated to research about, and promoting the use of, AD (http://www.coin-or.org/CppAD/Doc/cppad.htm). Figure 4.21 shows the sparse matrix of CppAD. A plot of eigenvalues of the bipartite matrix can be found in Figure 4.22. Figure 4.23 shows the architectural dependency graph of CppAD.

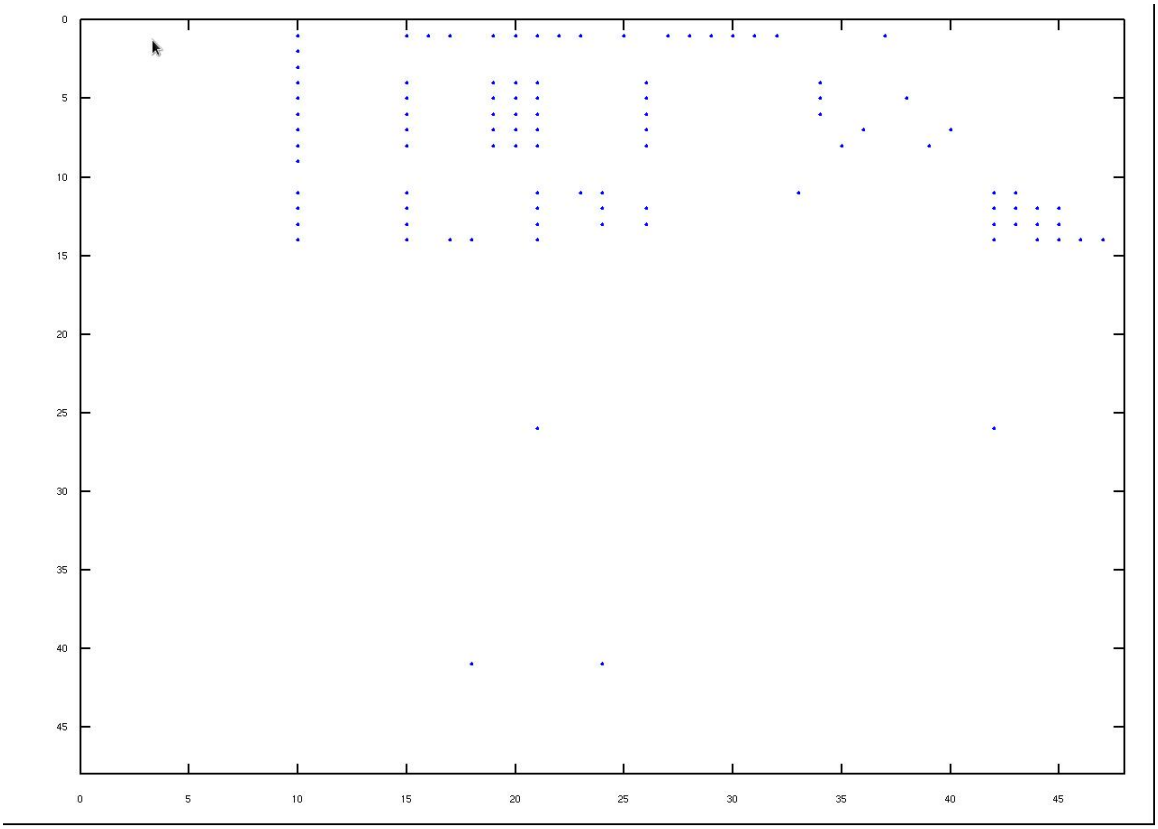


Figure 4.21: CppAD, sparse matrix

Figure 4.24 and Figure 4.25 displays the results obtained from applying HITS and Benzi's method on the non-weighted and weighted call graphs of CppAD. The ranking obtained by the method of Benzi et al. gives a similar picture as in the HITS ranking for the non-weighted call graph.

If we compare Benzi's method for the weighted and non-weighted call graph, the result is as follows:

Hub: The two functions, call() and operator[]*() are ranked number one and number two respectively for the non-weighted call graph. For the weighted call graph, the function operator[]*() and the function call() are ranked as number one and number two respectively.

Authority: In terms of non-weighted call graph, if we look at Figure 4.25, we can see some ambiguity for the functions hes_fg_map() and jac_g_map(). Both the functions have the same authority score and ranked as number three. Whereas, for weighted call graph, all the

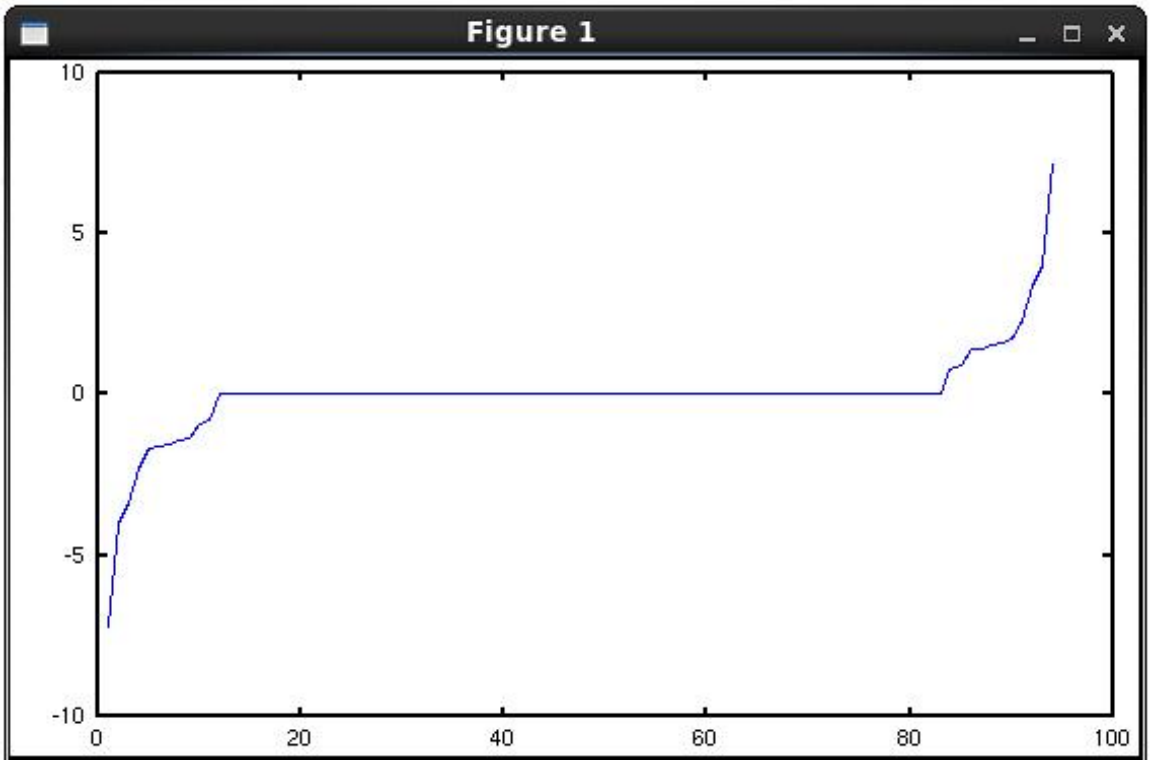


Figure 4.22: CppAD, graph plot of eigenvalues

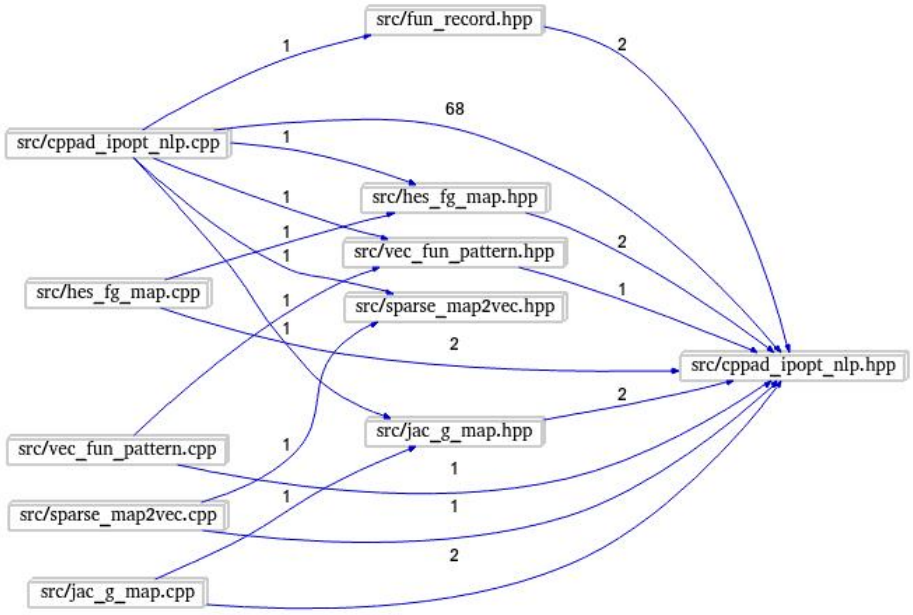


Figure 4.23: CppAD, dependency graph

five authority scores are distinct and hence the ranking is ambiguous.

Ranking using HITS Algorithm – CppAD

Function Name	*Hub rank*	*Indegree*	*Outdegree*
call()	0.45816	0	14
operator[]*()	0.44368	0	11
operator[]()	0.43184	0	10
index()	0.30982	2	7
operatorBool()	0.26917	0	6

Function Name	*Authority rank*	*Indegree*	*Outdegree*
cppad_ipopt_nlp()	0.34957	17	0
eval_grad_f()	0.32671	8	0
hes_fg_map()	0.32169	9	0
jac_g_map()	0.32169	9	0
eval_f()	0.32046	7	0

Ranking using HITS Algorithm (WEIGHTED) - CppAD

Function Name	*Hub rank*	*Indegree*	*Outdegree*
operator[]*()	0.91044	0	11
Call()	0.26373	0	14
operator[]()	0.21902	0	10
vector()	0.1149	0	2
operatorBool()	0.11172	0	6

Function Name	*Authority rank*	*Indegree*	*Outdegree*
cppad_ipopt_nlp()	0.50136	17	0
eval_h()	0.29002	8	0
eval_jac_g()	0.35731	8	0
vec_fun_pattern()	0.34633	10	0
eval_grad_f()	0.29002	8	0

Figure 4.24: CppAD, ranking using HITS algorithm

Ranking using Benzi's method – CppAD

Top 5 values

Hub rank	FUNCTION	INDEGREE	OUTDEGREE
164.7591	Call()	0	14
154.5075	operator[]*()	0	11
146.3724	operator[]()	0	10
75.3439	index()	2	7
56.8668	operatorBool()	0	6

Bottom 5 values

Hub rank	FUNCTION	INDEGREE	OUTDEGREE
4.36E-34	cppad_ipop_nlp()	17	0
9.04E-34	eval_jac_g()	8	0
3.18E-33	fun_record()	9	0
4.19E-33	eval_f()	6	0
7.52E-33	get_starting_point()	1	0

Top 5 values

Authority rank	FUNCTION	INDEGREE	OUTDEGREE
95.9161	cppad_ipopt_nlp()	17	0
83.7814	eval_grad_f()	8	0
81.2257	hes_fg_map()	9	0
81.22573	jac_g_map()	9	0
80.60341	eval_f()	7	0

Bottom 5 values

Authority rank	FUNCTION	INDEGREE	OUTDEGREE
1.10E-278	vector()	0	2
4.30E-219	domain_size()	0	1
2.70E-164	number_functions()	0	1
5.30E-136	number_terms()	0	1
4.20E-107	range_size()	0	1

Ranking using Benzi's method (WEIGHTED) – CppAD

Top 5 values

FUNCTION NAME	Hub rank	INDEGREE	OUTDEGREE
operator[]*()	5.58E+022	0	11
Call()	4.68E+021	0	14
operator[]()	3.23E+021	0	10
vector()	8.89E+020	0	2
operatorBool()	8.40E+020	0	6

Bottom 5 values

FUNCTION NAME	Hub rank	INDEGREE	OUTDEGREE
eval_h()	2.49E-014	8	0
jac_g_map()	3.00E-013	9	0
eval_f()	7.35E-012	7	0
eval_g()	5.71E-012	6	0
fun_record()	5.19E-012	9	0

Top 5 values

FUNCTION NAME	Authority Rank	INDEGREE	OUTDEGREE
cppad_ipopt_nlp()	1.69E+022	17	0
eval_h()	1.39E+022	8	0
eval_jac_g()	8.60E+021	8	0
vec_fun_pattern()	8.08E+021	10	0
eval_grad_f()	5.66E+021	8	0

Bottom 5 values

FUNCTION NAME	Authority Rank	INDEGREE	OUTDEGREE
vector()	5.76E-277	0	2
~vector()	2.41E-243	0	4
domain_size()	2.08E-209	0	1
number_functions	3.36E-145	0	1
number_terms()	4.53E-110	0	1

Figure 4.25: CppAD, ranking using benzi's method

Chapter 5

Summary and Future Work

In this thesis we have reviewed and applied ranking methods based on spectral analysis of the static call graphs portraying the relationship between caller and callee. We reviewed two sophisticated methods, HITS and method of Benzi et al. The DSM of the call graph provides a convenient tool so that linear algebraic techniques can be applied to identify important callers and callees through the calculation of matrix exponential. We applied these ranking methods on different sizes of networks. We tried to explore and extend the HITS and Benzi et al. methods by including the edge weights of the call graphs in determining the relative ranking of the function nodes.

We have used several networks for our experiments to demonstrate the effectiveness of Benzi et al. method relative to HITS and degree-based ranking. HITS and Benzi et al method was applied on both non-weighted call graph and weighted call graph. Our experiments imply that Benzi's method results in rankings that are often different from those computed by HITS. Our experiments also indicate that the the rankings obtained from Benzi et al method that was applied on non-weighted call graph differs from the rankings obtained from weighted call graph. As we have seen in Chapter 4, there are some ambiguity in the rankings obtained from non-weighted call graph, but later resolved, by producing all distinct values, when applied on weighted call graph. This is to be expected, since weighted call graph holds more information than non-weighted call graph.

In terms of further research, we would like to include more test problems on larger networks. It would be interesting to include problems from different scientific domain. Also,

in future, we would like to apply our method to software library of legacy code, where very little or no documentation is available about the code. The method that we developed in this thesis is likely to be useful for these types of library.

We would also like to work on other spectral techniques like *PageRank* and *reverse PageRank*.

Bibliography

[1] Stephen T Barnard, Alex Pothen, and Horst Simon. A spectral algorithm for envelope reduction of sparse matrices. *Numerical linear algebra with applications*, 2(4):317–334, 1995.

[2] Michele Benzi, Ernesto Estrada, and Christine Klymko. Ranking hubs and authorities using matrix functions. *Linear Algebra and its Applications*, 438(5):2447–2474, 2013.

[3] Phillip Bonacich. Factoring and weighting approaches to status scores and clique identification. *Journal of Mathematical Sociology*, 2(1):113–120, 1972.

[4] Stephen P Borgatti and Martin G Everett. A graph-theoretic perspective on centrality. *Social networks*, 28(4):466–484, 2006.

[5] Jonathan J Crofts and Desmond J Higham. A weighted communicability measure applied to complex brain networks. *Journal of the Royal Society Interface*, pages rsif–2008, 2009.

[6] Steven D Eppinger and Tyson R Browning. *Design structure matrix methods and applications*. MIT press, 2012.

[7] Ernesto Estrada. *The structure of complex networks: theory and applications*. Oxford University Press, 2011.

[8] Ernesto Estrada and Juan A Rodriguez-Velazquez. Subgraph centrality in complex networks. *Physical Review E*, 71(5):056103, 2005.

[9] Illes J Farkas, Imre Derényi, Albert-László Barabási, and Tamas Vicsek. Spectra of real-world graphs: Beyond the semicircle law. *Physical Review E*, 64(2):026704, 2001.

[10] Linton C Freeman. Centrality in social networks conceptual clarification. *Social networks*, 1(3):215–239, 1979.

[11] Jrg Feldhusen Karl-Heinrich Grote Gerhard Pahl, Wolfgang Beitz. *Engineering design: a systematic approach*. Springer London, 2007.

[12] Gene H Golub and Charles F Van Loan. *Matrix computations*, volume 3. JHU Press, 2012.

[13] Marco A Gonzalez. A new change propagation metric to assess software evolvability. 2013.

[14] Andreas Griewank, David Juedes, and Jean Utke. Adol-c: A package for the automatic differentiation. 1995.

[15] Shahadat Hossain, Ahmed Tahsin Zulkarnine, et al. Design structure of scientific software–a case study. 2011.

[16] Shahadat Hossain, Ahmed Tahsin Zulkarnine, et al. Design structure of scientific software–a case study. In *DSM 2011: Proceedings of the 13th International DSM Conference*, 2011.

[17] Shahadat Hossain, Ahmed Tahsin Zulkarnine, et al. Design structure of scientific software–a case study. In *DSM 2011: Proceedings of the 13th International DSM Conference*, 2011.

[18] Jon M Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM (JACM)*, 46(5):604–632, 1999.

[19] Matthew John LaMantia. *Dependency models as a basis for analyzing software product platform modularity: a case study in strategic software design rationalization*. PhD thesis, Massachusetts Institute of Technology, 2006.

[20] Anany V Levitin. *Introduction to Design & Analysis of Algorithms: For Anna University, 2/e*. Pearson Education India, 2009.

[21] Udo Lindemann, Maik Maurer, and Thomas Braun. *Structural complexity management: an approach for the field of product design*. Springer Science & Business Media, 2008.

[22] Gail C Murphy, David Notkin, William G Griswold, and Erica S Lan. An empirical study of static call graph extractors. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 7(2):158–191, 1998.

[23] S Unnikrishna Pillai, Torsten Suel, and Seunghun Cha. The perron-frobenius theorem: some of its applications. *Signal Processing Magazine, IEEE*, 22(2):62–75, 2005.

[24] Robert P Smith and Steven D Eppinger. Identifying controlling features of engineering design iteration. *Management Science*, 43(3):276–293, 1997.

[25] Gilbert Strang. *Introduction to Linear Algebra*. Wellesley Cambridge Press, 2003.

[26] Alexandru Telea, Hessel Hoogendorp, Ozan Ersoy, and Dennie Reniers. Extraction and visualization of call dependencies for large c/c++ code bases: A comparative study. In *Visualizing Software for Understanding and Analysis, 2009. VISSOFT 2009. 5th IEEE International Workshop on*, pages 81–88. IEEE, 2009.

[27] Alexandru Telea, Hessel Hoogendorp, Ozan Ersoy, and Dennie Reniers. Extraction and visualization of call dependencies for large c/c++ code bases: A comparative study. In *Visualizing Software for Understanding and Analysis, 2009. VISSOFT 2009. 5th IEEE International Workshop on*, pages 81–88. IEEE, 2009.

[28] Andrew H Tilstra, Matthew I Campbell, Kristin L Wood, Carolyn C Seepersad, et al. Comparing matrix-based and graph-based representations for product design. In *DSM 2010: Proceedings of the 12th International DSM Conference, Cambridge, UK, 22.-23.07. 2010*, 2010.

[29] Scientific Toolworks. Inc., understand for java: User guide and reference manual, 2005.

[30] Ahmed Tahsin Zulkarnine. Design structure and iterative release analysis of scientific software. 2012. [M.Sc Thesis].