

ELEMENT RANK

RANKING JAVA SOFTWARE CLASSES AND PACKAGES USING A MULTILAYER COMPLEX NETWORK-BASED APPROACH

Suresh (M20AIE313), Utkarsh (M20AIE318)

IIT Jodhpur

ABSTRACT

The software comprehension at the code level is a complex process as it involves the many layers, components, arch code level design structures, workflow, etc. Furthermore, the software API documentation, developer comments, and generating component diagrams may not help extensively understand the different software elements. Therefore, the comprehension process at the code level has a lot of complexities. In this work, we will show how the topological structure of software at the class level and package level. First, we use the ElementRank algorithm for ranking at class, and package level, then implement an analytic hierarchy process to weight each layer of multilayer software network and then implement the global weighted page rank for each class.

The outcome of the overall process would help the maintainers better understand code infrastructure in large software systems. In addition, quantifying the class importance with rank would help maintenance personnel with starting points of software comprehension at the code level. Finally, we will apply the proposed algorithm to various open-sourced projects and show how quantifying results help the maintainers.

Index Terms— PageRanking, Weighted Graph, Multi layer software network

1. INTRODUCTION

Understanding Large scale software systems with typical complex interacting elements like packages / classes / interfaces / inheritances / methods / attributes is a time taking process and cause maintenance of the code challenging task. The main objective is to provide the technology which would find the structure and importance of the classes and packages in large complex source code base so that maintenance personal time to understand the code would be reduced and help further to change the code structure better to improve the performance.

2. OBJECTIVES & SUMMARY

The Element ranking is enhanced formula of Page ranking to get the rank for each element and then aggregate with the Analytical hierarchy process. The weighted page rank value of each element class/package of any layer will be aggregated.

This process involves the steps:

- read the structure of the code and prepare the network layer graphs for each relation
- calculate the page rank of element and aggregating the weighted page rank with analytical hierarchy process

Relations found are:

- Inheritance INR
- Implements IMR.
- Parameter PAR
- Global variable GVR
- Method call MCR
- Local variable LVR
- Return type RTR

The relative importance between each relation is taken from the paper $W = (0.034, 0.290, 0.034, 0.034, 0.178, 0.394, 0.034)$

3. COMPARISON WITH EXISTING WORK

The actual page rank algorithm enhanced to consider the weighed graph as below

- Page Ranking

$$PR(p_i) = \frac{1-d}{N} + d \sum_{p_j \in \ln Links(p_i)} \frac{PR(p_j)}{OutLinks(p_j)} \quad (1)$$

- Weighted Page Rank

$$PR_w(p_i) = \frac{(1-d) \times w \ln Links(p_i)}{N} + d \sum_{k=1}^N w \ln Links(p_k) + d \sum_{p_j \in \ln Links(p_i)} \frac{PR_w(p_j \times w(p_j, p_i))}{w OutLinks(p_j)} \quad (2)$$


- Aggregated or Global weighted page rank

$$PR_w^g(p_i) = \sum_{l=1}^o w_l \times PR_w^l(p_i) \quad (3)$$

4. TESTING

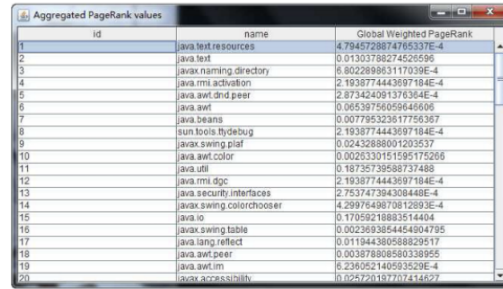
As part of algorithm testing, we have taken the java jdk code MPN network to compute the global weighted page ranks.

4.1. Ranks calculated by our code



Package	Rank
java.text.resources	4.794573E-4
java.text	0.013037883
javax.naming.directory	6.80229E-4
java.rmi.activation	2.1938774E-4
java.awt.dnd.peer	2.873424E-4
java.awt	0.06539756
java.beans	0.0077953236
sun.tools.ttydebug	2.1938774E-4
javax.swing.plaf	0.02432888
java.awt.color	0.0026330152
java.util	0.1873574
java.rmi.dgc	2.1938774E-4
java.security.interfaces	2.7537474E-4
javax.swing.colorchooser	4.299765E-4
java.io	0.17059219
javax.swing.table	0.0023693854
java.lang.reflect	0.011944381
java.awt.peer	0.0038788086
java.awt.im	6.236052E-4
javax.accessibility	0.025720198
java.awt.event	0.034040112
javax.swing.border	0.015758822
java.sql	2.1938774E-4

4.2. original ranks provided for the same data set from paper



id	name	Global Weighted PageRank
1	java.text.resources	4.7945728874765337E-4
2	java.text	0.01303788274526596
3	javax.naming.directory	6.80228983117039E-4
4	java.rmi.activation	2.1938774443697184E-4
5	java.awt.dnd.peer	2.873424091376364E-4
6	java.awt	0.06539756059646605
7	java.beans	0.007795323617756367
8	sun.tools.ttydebug	2.1938774443697184E-4
9	javax.swing.plaf	0.02432888001203537
10	java.awt.color	0.0026330151595175266
11	java.util	0.18735739588737488
12	java.rmi.dgc	2.1938774443697184E-4
13	java.security.interfaces	2.753747394308448E-4
14	javax.swing.colorchooser	4.2997649870812893E-4
15	java.io	0.17059218803514404
16	javax.swing.table	0.0023693854464004795
17	java.lang.reflect	0.011944380588829517
18	java.awt.peer	0.00387880860338955
19	java.awt.im	6.236052140503529E-4
20	javax.accessibility	0.025720187207444697

(b) Package important list

5. LINKS

- GitHub Code
- Demo Video

6. SUMMARY

In this project we have successfully extracted classes and other packages from java jdk code, applied the algorithm suggested in the paper and have received identical efficiency as that of the paper.