Project Title: System Verification and Validation Plan for Centrality in Graphs

Atiyeh Sayadi

March 7, 2024

Revision History

Date	Version	Notes
Feburary 14, 2024	1.0	Initial draft
Date 2	1.1	Notes

Contents

List of Tables

[Remove this section if it isn't needed —SS]

List of Figures

[Remove this section if it isn't needed —SS]

1 Symbols, Abbreviations, and Acronyms

symbol	description
CC	Closeness Centrality
CD	Degree Centrality
SRS	Software Requirment Specification
MG	Module Guide
MIS	Module Interface Specification
VnV	Verification and Validation

This section includes symbols and symbols used for developing this document.

This document is created to review and confirm the project from its early stages of project clarification to design and implementation. It will cover various aspects, including setting up documents and using different techniques and tools for testing the project.

2 General Information

2.1 Summary

We want to design software that calculates closeness centrality and degree centrality for each node in a given graph, represented in the form of a numerical matrix of nodes and edges. Since this software is computational in nature, the accuracy and performance of functions and modules, including the main modules that calculate closeness centrality and degree centrality, are crucial. Therefore, it is necessary for this software to be tested and validated from various aspects. This document serves as a roadmap for testing this software from pre-development to post-development stages.

2.2 Objectives

The purpose of developing this document is to accurately measure the correctness and performance of modules and the overall software, and to verify the software. For example, at the end and after generating the output, it is necessary to know how valid and accurate the produced results are. To achieve this, we can use library functions in Python that perform these calculations and compare the software output with them. Additionally, this document will not cover all aspects of software testing because, for instance, since we are developing a simple user interface for the software, usability testing is not relevant.

2.3 Relevant Documentation

At this stage of document setup, other documents such as the Software Requirements Specification (SRS), the MIS ans TH MG have been utilized. This is because at this stage, it is necessary to specify the development objectives, requirements, assumptions, and constraints of the software, and accordingly, testing is conducted. Additionally, precise computational algorithms need to

be identified to assess their accuracy and precision. All of these aspects are summarized in these documents.

3 Plan

The purpose of this section is to identify and examine various aspects and relevant items related to this project for testing. Therefore, documents such as the SRS, the MIS and the MG need to be reviewed and verified from various perspectives.

3.1 Verification and Validation Team

This software is being developed individually. Therefore, initially, it is the developer's responsibility to ensure the correctness of its functioning and a proper understanding of the requirements. Additionally, reviewers can inform about any issues encountered during each stage of development.

	SRS	**VnV**	**MIS/MG**
Atiyeh Sayadi	X	X	X
Yiding Li	X	X	X
Tanya Djavaherpour	X		
Cynthia Liu		X	
Waqar Awan			X
Dr.Spencer Smith	X	X	X

Table 1: The table represents the responsibility of each team member.

3.2 SRS Verification Plan

The SRS document can be verified by reviewers according to the following:
1- Confirmation of project requirements and assumptions.
2- Confirmation of project objectives aligned with requirements.
3- Confirmation of the clarity in explaining algorithms, their concepts, and their applicability.

Process: The document is uploaded to GitHub. Relevant reviewers examine it and provide their feedback in the form of issues.

3.3 Design Verification Plan

The design phase is an intermediate and critical stage where the final software is designed based on features, requirements, and assumptions. In this section, the main functions and modules are designed. Therefore, reviewers need to examine whether this project has been designed practically and correctly in accordance with its objectives and requirements.

Process: The document is uploaded to GitHub. Relevant reviewers examine it and provide their feedback in the form of issues.

3.4 Verification and Validation Plan Verification Plan

In software development, all stages, including software testing, require verification. In this regard, it is necessary to examine whether the tests planned for the software are sufficient or not. Additionally, sometimes some tests are overly complex or impractical to execute, and they need to be identified and removed from the verification process. This confirmation is carried out by reviewers.

Process: The document is uploaded to GitHub. Relevant reviewers examine it and provide their feedback in the form of issues.

3.5 Implementation Verification Plan

In the final stage, the software needs to be approved. This approval involves:
1. Verification of whether the software has been developed according to the requirements, goals, algorithms, and assumptions specified in the SRS document or not.
2. Checking whether the final software conforms to the design documents from various aspects or not.
3. Ensuring whether the output is clear or not.

3.6 Automated Testing and Verification Tools

As mentioned, this software will primarily be tested for accuracy and performance. Additionally, the focus will be mainly on unit testing and system testing. Therefore, it is likely that pytest will be used for testing this software. Furthermore, for addressing issues related to coding errors and standardizing them, pylint will be utilized.

3.7 Software Validation Plan

To validate any software, it is necessary to first verify it from various aspects. After the software has been verified, the project owner needs to validate it. This project will be validated by Dr. Spencer Smith.

4 System Test Description

4.1 Tests for Functional Requirements

In this section, the specific items to be tested in the software are described in detail.

4.1.1 Degree Centrality Testing

In this section, the outputs related to degree centrality are being tested.

1. Test1: Bound Testing
   Control: Automatic
   Initial State: null

   Input: The outputs of degree centrality calculations.

   Output: nodes that their values of cenrality are out of bound.

   Test Case Derivation: A function should be written to demonstrate that the output for each node is within the range of zero to one.

   How test will be performed: It dermindes nodes that their centrality are less than 0 or more than 1.

2. Test2: Node Testing

Control: Automatic

Initial State: null

Input: The outputs of centrality degree calculations.

Output: Nodes for which the degree centrality has not been calculated.

Test Case Derivation: A function should be written as a test function to demonstrate that centrality has been computed for all nodes present in the graph.

How test will be performed: Nodes for which centrality has not been calculated are displayed.

4.1.2 Closeness Centrality Testing

In this section, the outputs related to calculating closeness centrality are being tested.

1. Test3: Bound Testing
   Control: Automatic
   Initial State: null

   Input: The outputs of degree centrality calculations.

   Output: nodes that their values of cenrality are out of bound

   Test Case Derivation: A function should be written to demonstrate that the output for each node is within the range of zero to one.

   How test will be performed: It dermindes nodes that their centrality are less than 0 or more than 1.

2. Test4: Node Testing

   Control: Automatic

   Initial State: null

   Input: The outputs of centrality degree calculations.

   Output: Nodes for which the closenedd centrality has not been calculated.

Test Case Derivation: A function should be written as a test function to demonstrate that centrality has been computed for all nodes present in the graph.

How test will be performed: Nodes for which centrality has not been calculated are displayed.

4.2 Tests for Nonfunctional Requirements

Since resources and time for developing this software are limited, the only test conducted in this regard is maintainability and accuracy testing.

4.2.1 maintainability

1. Test5: Code Testing

   Type: Automatic

   Initial State: -

   Input/Condition: Program code

   Output/Result: Coding errors

   Test Case Derivation: Using the pylint module

   How test will be performed: Highlighted are the areas where programming standards have not been followed, with explanations provided in the terminal.

4.2.2 Accuracy

1. Test6: Correctness Testing for DC

   Control: Automatic

   Initial State: null

   Input: The outputs of degree centrality calculations.

   Output: Nodes whose centrality contradicts what is calculated by library functions for this criterion.

Test Case Derivation: Using the networkx library functions in Python, the centrality of each node is calculated for the given graph, and it is compared with the output of the program.

How test will be performed: Nodes for which the calculated centrality differs from the library function output are displayed.

2. Test7: Correctness Testing for CC

   Control: Automatic

   Initial State: null

   Input: The outputs of closeness centrality calculations.

   Output: Nodes whose centrality contradicts what is calculated by library functions for this criterion.

   Test Case Derivation: Using the networkx library functions in Python, the centrality of each node is calculated for the given graph, and it is compared with the output of the program.

   How test will be performed: Nodes for which the calculated centrality differs from the library function output are displayed.

4.3 Traceability Between Test Cases and Requirements

5 Unit Test Description

[This section should not be filled in until after the MIS (detailed design document) has been completed. —SS]

[Reference your MIS (detailed design document) and explain your overall philosophy for test case selection. —SS]

[To save space and time, it may be an option to provide less detail in this section. For the unit tests you can potentially layout your testing strategy here. That is, you can explain how tests will be selected for each module. For instance, your test building approach could be test cases for each access program, including one test for normal behaviour and as many tests as needed for edge cases. Rather than create the details of the input and output here,

	FR1	FR2	FR3	FR4	NFR1	NFR2	NFR3	NFR4
Test 1	X				X			
Test 2		X			X			
Test 3			X		X			
Test 4				X	X			
Test 5							X	
Test 6					X			
Test 7					X			

Table 2: Traceability Matrix Showing the Connections Between Tests and Requirments in SRS

you could point to the unit testing code. For this to work, you code needs to be well-documented, with meaningful names for all of the tests. —SS]

5.1 Unit Testing Scope

[What modules are outside of the scope. If there are modules that are developed by someone else, then you would say here if you aren't planning on verifying them. There may also be modules that are part of your software, but have a lower priority for verification than others. If this is the case, explain your rationale for the ranking of module importance. —SS]

5.2 Tests for Functional Requirements

[Most of the verification will be through automated unit testing. If appropriate specific modules can be verified by a non-testing based technique. That can also be documented in this section. —SS]

5.2.1 Module 1

[Include a blurb here to explain why the subsections below cover the module. References to the MIS would be good. You will want tests from a black box perspective and from a white box perspective. Explain to the reader how the tests were selected. —SS]

1. test-id1

   Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

   Initial State:

   Input:

   Output: [The expected result for the given inputs —SS]

   Test Case Derivation: [Justify the expected value given in the Output field —SS]

   How test will be performed:

2. test-id2

   Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

   Initial State:

   Input:

   Output: [The expected result for the given inputs —SS]

   Test Case Derivation: [Justify the expected value given in the Output field —SS]

   How test will be performed:

3. ...

5.2.2 Module 2

...

5.3 Tests for Nonfunctional Requirements

[If there is a module that needs to be independently assessed for performance, those test cases can go here. In some projects, planning for nonfunctional tests of units will not be that relevant. —SS]

[These tests may involve collecting performance data from previously mentioned functional tests. —SS]

5.3.1 Module ?

1. test-id1

   Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

   Initial State:

   Input/Condition:

   Output/Result:

   How test will be performed:

2. test-id2

   Type: Functional, Dynamic, Manual, Static etc.

   Initial State:

   Input:

   Output:

   How test will be performed:

5.3.2 Module ?

...

5.4 Traceability Between Test Cases and Modules

[Provide evidence that all of the modules have been considered. —SS]

6 Appendix

As previously mentioned, this document aims to provide information about testing a software project from start to finish, covering different methods and aspects. Therefore, this document will assist in achieving a more precise and targeted testing and validation process.

6.1 Symbolic Parameters

In the development of this document, no symbolic constants have been used.

6.2 Usability Survey Questions?

Since the development of a graphical interface is not part of the development agenda for this project, no description can be considered for this section.

Appendix — Reflection

The information in this section will be used to evaluate the team members on the graduate attribute of Lifelong Learning. Please answer the following questions:

Appendix — Reflection

[This section is not required for CAS 741 —SS]

The information in this section will be used to evaluate the team members on the graduate attribute of Lifelong Learning. Please answer the following questions:

1. What knowledge and skills will the team collectively need to acquire to successfully complete the verification and validation of your project? Examples of possible knowledge and skills include dynamic testing knowledge, static testing knowledge, specific tool usage etc. You should look to identify at least one item for each team member.

2. For each of the knowledge areas and skills identified in the previous question, what are at least two approaches to acquiring the knowledge or mastering the skill? Of the identified approaches, which will each team member pursue, and why did they make this choice?