System Verification and Validation Plan for Centrality in Graphs

Atiyeh Sayadi

April 1, 2024

Revision History

Date	Version	Notes
Feburary 14, 2024	1.0	Initial draft
March 22, 2024	1.1	Second review
April 1, 2024	1.2	Unit testing

Contents

List of Tables

List of Figures

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

In this section, a brief overview of the project and its dimensions for testing purposes, as well as the purpose of writing this document and the documents used in drafting this project, are provided.

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 ensure the accuracy and reliability of the project from various aspects. Accordingly, it is necessary for the project to be tested from different perspectives and stages based on the defined requirements. Since the functional requirements of the final system are defined based on the accuracy and precision of the outputs, this issue encompasses a significant portion of the tests. Additionally, since this system will provide a simple user interface, it does not seem necessary to perform usability testing for non-functional requirements. Furthermore, due to time constraints, it is not feasible to install the software on various systems to fully assess its portability.

2.3 Relevant Documentation

At this stage of document setup, other documents such as the Software Requirements Specification (SRS), the MIS and 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. In the first stage, it is necessary to identify which individuals are involved in testing this project and what each person's role is. Then, the Software Requirements Specification (SRS) document is reviewed as the first important document for testing this software. Following that, the testing procedure for design documents, including even this test document, is examined, and then the final software testing is introduced. Since testing is carried out both automatically and manually, automated testing tools will also be introduced.

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. The list of individuals and their responsibilities is provided in Table 1.

Reviewers	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.
In this stage, besides the review conducted by the Dr.Spencer Smith, it is necessary to develop automated tests to confirm the accuracy of the software from various aspects.

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, 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: Based on requirements 1 and 2 of this project, it is necessary for the degree centrality to be accurately calculated for all nodes. We know that this metric for each node is between zero and one. Therefore, a function needs to be written to evaluate this value for each node based on the allowable range and report any discrepancies found.

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

4.1.2 Closeness Centrality Testing

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

1. Test2: 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: Based on requirements 3 and 4 of this project, it is necessary for the closeness centrality to be accurately calculated for

all nodes. We know that this metric for each node is between zero and one. Therefore, a function needs to be written to evaluate this value for each node based on the allowable range and report any discrepancies found.

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

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. Test3: 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. Test4: 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(Based on requirements 1 and 2).

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

2. Test5: 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(Based on requirements 3 and 4).

   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

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

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

5 Unit Test Description

In this section, various software modules and the performance of their functions are tested based on the MIS and MG documents. According to these two documents, this software consists of three main modules: File, Graph, and ShowGraph, with the primary computational burden of this software falling on the Graph module. Since the other two modules do not perform primary computations, functional requirements are tested only for the Graph module. Typically, with the use of automated tests written in pytest, the output for degree centrality, closeness centrality, shortest path length, and node degree is tested for accuracy using the networkx library, which determines these outputs.

5.1 Unit Testing Scope

As mentioned earlier, tests are mainly focused on the Graph module, which carries the main computational load and ensures the correct functionality of the program.

5.2 Tests for Functional Requirements

In this section, the tests considered for each module are titled according to functional requirements.

5.2.1 Graph Module

As previously mentioned, the main computations of this software are performed by this module, which includes functions such as computing degree centrality, closeness centrality, node degree, and shortest path length between two nodes. Therefore, each of these functions can be separately examined for accuracy and correctness to ensure the validity of the software.

1. degree_centrality_bound

   Type: Automatic

   Initial State: nodes=[], A list of nodes whose degree centrality is outside the specified range is empty.

   Input: -

Output: Assertion of nodes whose degree centrality is outside the specified range. If the list is empty, the test will pass.

Test Case Derivation: Nodes that have degree centrality less than zero or greater than one.

How test will be performed: The degree centrality for each node should fall between zero and one. Therefore, in the body of this function, the get_degree_centrality function is called, and the degree centrality for each node is calculated to determine whether it falls within the specified range or not.

2. closeness_centrality_bound

   Type: Automatic

   Initial State: nodes=[], A list of nodes whose closeness centrality is outside the specified range is empty.

   Input: -

   Output: Assertion of nodes whose closeness centrality is outside the specified range. If the list is empty, the test will pass.

   Test Case Derivation: Nodes that have closeness centrality less than zero or greater than one.

   How test will be performed: The closeness centrality for each node should fall between zero and one. Therefore, in the body of this function, the get_closeness_centrality function is called, and the closeness centrality for each node is calculated to determine whether it falls within the specified range or not.

3. degree_centrality_values

   Type: Automatic

   Initial State: nodes=[], A list of nodes whose degree centrality deviates from the expected value.

   Input: -

   Output: Assertion of nodes whose degree centrality deviates from the output of the NetworkX library. If the list is empty, the test will pass.

Test Case Derivation: The degree centrality of these nodes deviates from the output of the NetworkX library.

How test will be performed: The function get_degree_centrality is called within the body, and its value is computed for each node by comparing it with the value of the degree_centrality function from the NetworkX library. If there is a discrepancy in the values, the node is added to the list of nodes.

4. closeness_centrality_values

   Type: Automatic

   Initial State: nodes=[], A list of nodes whose closeness centrality deviates from the expected value.

   Input: -

   Output: Assertion of nodes whose closeness centrality deviates from the output of the NetworkX library. If the list is empty, the test will pass.

   Test Case Derivation: The degree centrality of these nodes deviates from the output of the NetworkX library.

   How test will be performed: The function get_closeness_centrality is called within the body, and its value is computed for each node by comparing it with the value of the closeness_centrality function from the NetworkX library. If there is a discrepancy in the values, the node is added to the list of nodes.

5. degree_values

   Type: Automatic

   Initial State: nodes=[], A list of nodes whose degree deviates from the expected value.

   Input: -

   Output: Assertion of nodes whose degree deviates from the output of the NetworkX library. If the list is empty, the test will pass.

   Test Case Derivation: The degree of these nodes deviates from the output of the NetworkX library.

How test will be performed: The function get_degree is called within the body, and its value is computed for each node by comparing it with the value of the degree function from the NetworkX library. If there is a discrepancy in the values, the node is added to the list of nodes.

6. shortest_path

   Type: Automatic

   Initial State: nodes=[], A list of nodes whose length of shortest path to other nodes deviates from the expected value.

   Input: -

   Output: Assertion of nodes whose length of shortest path deviates from the output of the NetworkX library. If the list is empty, the test will pass.

   Test Case Derivation: The length of shortest path of these nodes deviates from the output of the NetworkX library.

   How test will be performed: The function get_shortest_path is called within the body, and its value is computed for each node by comparing it with the value of the shortest_path function from the NetworkX library. If there is a discrepancy in the values, the node is added to the list of nodes.

5.3 Tests for Nonfunctional Requirements

In addition to the functional requirements, it is necessary to test the modules based on non-functional requirements as well. Since one of these requirements is the accuracy of the output, this will be examined based on the tests performed for the graph module in the previous section. Additionally, since a simple graphical interface is considered for this software, implemented by the ShowGraph module, usability testing for it is not of great importance. Furthermore, in designing this software, an attempt has been made to utilize a hierarchical modular design approach with minimal side effects, which can serve as confirmation of its maintainability. The only test considered in this section is the accuracy test of the software's performance based on various inputs, which involves all modules.

1. Input Testing

   Type: Automatic

   Initial State: -

   Input/Condition: Different Input files

   Output/Result: The module's performance for different inputs varies.

   How test will be performed: We know that the software is designed for undirected, connected, and unweighted graphs. In this test, various types of inputs, including inputs that violate the project's assumptions, are provided to the system to determine its performance based on these inputs.

5.4 Traceability Between Test Cases and Modules

	File	Graph	ShowGraph
Test 1		X	
Test 2		X	
Test 3		X	
Test 4		X	
Test 5		X	
Test 6		X	
Test 7	X	X	X

Table 3: Traceability Matrix Showing the Connections Between Tests and Modules

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?