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

Atiyeh Sayadi

February 11, 2024

Revision History

Date	Version	Notes
Feburary 11, 2024	1.0	Notes
Date 2	1.1	Notes

[The intention of the VnV plan is to increase confidence in the software. However, this does not mean listing every verification and validation technique that has ever been devised. The VnV plan should also be a **feasible** plan. Execution of the plan should be possible with the time and team available. If the full plan cannot be completed during the time available, it can either be modified to "fake it", or a better solution is to add a section describing what work has been completed and what work is still planned for the future. —SS]

[The VnV plan is typically started after the requirements stage, but before the design stage. This means that the sections related to unit testing cannot initially be completed. The sections will be filled in after the design stage is complete. the final version of the VnV plan should have all sections filled in. —SS]

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	Degree Centrality
CD	Closeness Centrality

The table that follows summarizes the symbols used in this document along with their units. The choice of symbols was made to be consistent with the heat transfer literature and with existing documentation for solar water heating systems. The symbols are listed in alphabetical order.

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 two 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 not developing a 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) and the Problem Statement 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 the Problem Statement document and elaborated more precisely in the SRS.

Author (2019)

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, designs, and ultimately the final software need to be reviewed and validated 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.

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.

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. The design phase of this project can only be approved by those who are aware of the dimensions and objectives of this software. Therefore, reviewers need to examine whether this project has been designed practically and correctly in accordance with its objectives and requirements.

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.

3.5 Implementation Verification Plan

The developed software needs to be tested in the following areas:
1- Input file format: Ensuring that the input file is formatted according to the standard for reading.
2- Closeness centrality output: The output for closeness centrality for each node should range between zero and one.
3- Degree centrality output: The output for degree centrality for each node should range between zero and one.
Therefore, it is necessary to manually test the above cases with code development. Additionally, Python libraries that calculate closeness centrality and degree centrality for graphs can be used to ensure that the program's output is accurate and precise. Furthermore, by changing the input dataset, it can be ensured that the software is usable for all standard datasets.

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

1- The input file must be a numerical matrix with only two columns.
2- The outputs should include calculations for closeness centrality and degree centrality for all nodes.

3- The outputs for closeness centrality and degree centrality should be between zero and one.
In addition to the mentioned aspects, this project is supervised by reviewers. Therefore, all aspects, including software testing, need to be verified and any existing issues should be identified by them.

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 Area of Testing1

In this section, it will be determined how the input of the program should be tested for conformity to standards.

Input Testing

1. test-id1

   Control: Manual

   Initial State: (0*0)

   Input: Input Matrix

   Output: The size of the Matrix

   Test Case Derivation: size: (n*2)

   How test will be performed: by using of the numpy library to calculate the size.

4.1.2 Area of Testing2

In this section, the output of the degree centrality calculation module is tested Control: Automatic

1. test-id1
   Control: Manual Initial State: null

   Input: the out put of the function of DC (Degree Centrality)

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

   Test Case Derivation: a function should be developed

   How test will be performed: it dermindes the node that their centrality are less than 0 or more than 1.

2. test-id2

   Control: Manual

   Initial State: null

   Input: the out put of the function of DC (Degree Centrality)

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

   Test Case Derivation: a function should be developed

3. test-id3

   Control: Manual

   Initial State: null

   Input: the out put of the function of DC (Degree Centrality)

   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.

4.1.3 Area of Testing3

In this section, the output of the closeness centrality calculation module is tested Control: Automatic

1. test-id1
   Control: Manual Initial State: null

Input: the out put of the function of CC (Closeness Centrality)

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

Test Case Derivation: a function should be developed

How test will be performed: it dermindes the node that their centrality are less than 0 or more than 1.

2. test-id2

   Control: Manual

   Initial State: null

   Input: the out put of the function of CC (Closeness Centrality)

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

   Test Case Derivation: a function should be developed

3. test-id3

   Control: Manual

   Initial State: null

   Input: the out put of the function of CC (Closeness Centrality)

   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.

4.2 Tests for Nonfunctional Requirements

Since resources and time for developing this software are limited, the only test conducted in this regard is documentation testing, which is performed by reviewers. They then provide their feedback in written form for further adjustments.

4.2.1 Documents

Document Testing

1. test-id1

   Type: Manual

   Initial State: -

   Input/Condition: All documents shoulf be uploaded.

   Output/Result: Reviewers' comments on the documentation.

   How test will be performed: At least three reviewers read the documentation and provide feedback on any shortcomings.

4.3 Traceability Between Test Cases and Requirements

	R1	R2	R3	Accuracy
Test 1	X			X
Test 2		X		X
Test 3			X	X

Table 1: Traceability Matrix Showing the Connections Between Tests and Requirments in SRS

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, 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]

References

Author Author. System requirements specification. https://github.com/..., 2019.

6 Appendix

This is where you can place additional information.

6.1 Symbolic Parameters

The definition of the test cases will call for SYMBOLIC_CONSTANTS. Their values are defined in this section for easy maintenance.

6.2 Usability Survey Questions?

[This is a section that would be appropriate for some projects. —SS]

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?