Requirement Document for [SEP7] Group

1 Introduction

Brad Schoone, Dani Derviskadic, Josh Batchelor and Paul Fraser will be designing software for Johannes Herrmann. The software will use the OBDD-H algorithm to calculate the reliability of a network as described in Dr Herrmann's thesis, "Efficient Calculation of Reliability and Performability of Large Computer Networks". Mark Upston will be the supervisor for the duration of this project.

2 System

As suggested, the algorithm will be written in C++ for it's efficiency, extensibility and modularity. The user interface will be coded in Java, as opposed to C#. Our reason for this decision is because our intention is to have this program cross platform compatible, something that Java would enable us to do over C# which is currently Windows OS based. Java provides both the modularity and functionality that can be achieved in C#.

Our intention is to also rewrite the Perl scripts provided to us in Python. This will increase maintainability, readability and modularity. While Perl is very powerful we feel that in the long term Python will be the better option.

All of the code that we produce and write, along with all documentation, meeting minutes and other important information will be stored on the BitBucket repository provided to us by Johannes Herrmann, and will be published under a license of Johannes' preference.

For our project management solution, we have decided to use a combination of both Trello and Excel. Our product backlog, task allocations and current sprint log will be recorded and updated in Trello. An excel spreadsheet will be set up to record our time allocations for each task, our actual time allocations for each task and other relevant statistics on our task timings.

3 Functional requirements

3.1) The program must be able to be launched into a graphical user interface.

3.2) The graphical user interface must be able to load an existing, sorted network file, generate new network files and save network files.

3.3) The graphical user interface must display the end reliability of the network in decimal format, the number of nodes generated in the whole tree, the number of total nodes processed and the maximum number nodes at any one level.

3.4) The software must be able to calculate the reliability of a network, using the OBDD-H algorithm provided by the client, with as low of a time complexity as possible. This involves using binary encoding, and storing nodes in a hash table.

3.6) The software must have a toggleable ability to write the results as stated in (3.8) to a log file for later use/analysis, and must be human readable.

3.7) The software will be able to load a sorted network file for computation, in the format supplied by Johannes' examples.

3.8) The software will compute the reliability of a network being successful using the OBDD-H algorithm and output the reliability, the time taken to complete, as well as maximum number of nodes at a level, total number of nodes generated in the whole tree and the total number of nodes processed.

3.9) Must have a User Manual and API documentation suitable for a user interacting with the system and a developer interpreting the system.

4 Non-functional requirements

4.1) The results displayed or saved to a log file will be accurate and correct.

4.2) The results displayed or saved should be consistent for the same network.

4.3) Execution of the the OBDD-H algorithm will complete in a time no slower than those provided to us from previous tests.

4.4) The code will adhere to Curtin University's Coding Standard.

4.4) Design and code a separate brute-force application to accurately calculate the reliability of networks as a comparison to the results generated by this (OBDD-H) program.

5 Constraints

5.1) We will not be able to develop the interface for the software with C# or conduct verification testing on Windows. This is due to highly limited access to machines with both the Windows operating system and the required software installed.

5.2) The network files supplied must be ordered when they are loaded. They must first be ordered by the second column (vertex) and have ties resolved by ordering the third column (adjoining vertex).

5.3) The edges supplied to this algorithm must be bidirectional. If the input is invalid, an error message must be displayed explaining that the algorithm is made to work for bidirectional graphs only. A graph with a bidirectional edge will not be loaded.

6 Verification

6.1) Verification must be done on calculated networks for correctness and maintainability

6.2) Consistently receive the same reliability result as the accurate brute force application as stated in (4.4).

6.3) Must be compared to previous algorithm times, to make sure its running time is less than times previously recorded.

6.4) A 20 Vertex, fully connected graph is the most ideal test case of functionality.

6.5) The probabilities of reliability and unreliability must end up equaling a probability of 1 ad the end of the calculation.

7 Extensions

7.1) Writing the journal article for publication.

7.2) The ability to queue and run multiple networks one after the other, or concurrently.

7.3) Calculating the average time of running the same network multiple times.

7.4) Displaying the topology of the network.

7.5) Allow the user to supply an input file which specifies a K-REL group instead of a SOURCE and SINK node specified in the original graph file.

7.6) Optimise the software for multithreaded and/or distributed computing

7.7) Enable hardware accelerated processing in the GPU.

7.8) Allow the user to supply an input file which specifies an ALL-REL graph problem.