CSED

Signal Flow Graph Solver

Ahmad Abdallah Waheeb (04)

Ahmed Mohamed El Zeny (06)

April 08, 2019

Problem Statement

Given The total number of nodes and the numeric branches gains, it's required to draw the graph, print the forward paths, individual loop gains, combinations loop gains, deltas, and Overall system transfer Function by applying Mason's Rule.

Main Features of Program

1. **Excellent UX:** the program allows the user to build the graph just by drag and drop, and some simple techniques.
2. **Database insurance:** the program was done using node.js, Express.js and Jquery so as long as the server is running the data is safe, also the user can refresh the page if the user faced any error.
3. **Graph manipulation:** the user can delete nodes and edges while building the graph.
4. **Supports all kind of gains:** the program supports numeric and non numeric branches gains.

Data Structures

We used 2 major libraries to represent the graph structure:

1. Vis.js is a graph visualization library that uses nodes and edges storing ids and labels for nodes and ids, to-node and from-node for edges.
2. Graphlib, A directed multi-graph library for JavaScript we used it for logic, it has the same data structures as in vis so we stores them in the database (Jquery) and refilling them when opening the page.
3. ForwardPathes[ ][ ] a 2d array containing the different paths each contains the ids of the nodes.
4. Loops [ ] [ ] a 2d array containing different loops in the graph each contains the ids of the nodes
5. UnTouchedLoops[ ] [ ] [ ] a 3d array of the same length as ForwardPathes that contains the loops that doesn't touch that certain path.
6. nonTouchinLoops [ ] [ ] [ ] a 3d of the same length as the loops array. First array indicates the number of the non touched loops array. The second indicates a group (array) of non touching loops. The third is the loop index.

Main modules

We have 3 main modules

1. index.HTML contains the front End vis-related functions like add, delete nodes and also talks to the database exporting the nodes and edges data.
2. index.Js contains the server listening function (Express.js) and the server data creation and storing
3. Additional.js contains all the logic-related methods and data structures like getForwardPaths() and getDelta(), also contains some printing-to-HTML methodes.

Algorithms used

Function `forwardPaths(id)`

This function gets all the forward paths from the input node to the sink. It uses recursion and a stack to recover each path and pushes it to the paths[] array. This function also finds all the loops in the graph and pushes each loop to the loops[] array .

Function `getNonTouching(loops)`

This function finds all combinations of non touching loops. It starts by finding combinations of 2 non touching loops. Then it uses these groups to find the combinations of 3 non touching loops and so on.

Function `getDelta(loops)`

This function gets Delta of Mason's Rule by first calling the `getNonTouching(loops)` which fills the nonTouchinLoops[][][] array and then combining them by this formula :

Δ = 1 - (sum of all loops gain) + (each 2 non touching loops multiplied) - (each 3 non touching loops multiplied) and so on.

Function `removeTouched()`

Used for removing the loops that touch a certain path, it Loops over the paths array compares every path of them to the loops in the loops array then filters out the loops that contains a touching node to the path.

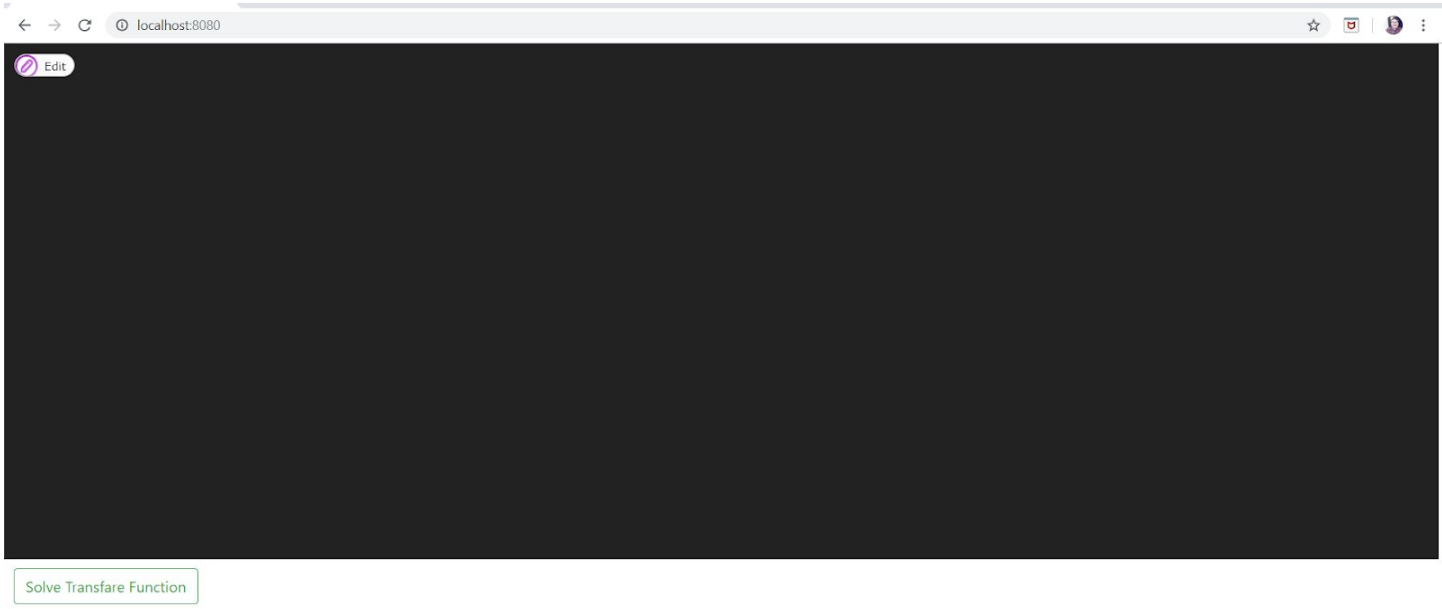
Function `getDeltas()`

This function uses the previous one to filter out the touched loops and pass these loops th the function getDelta, it also prints out the deltas functions to the Html Page.

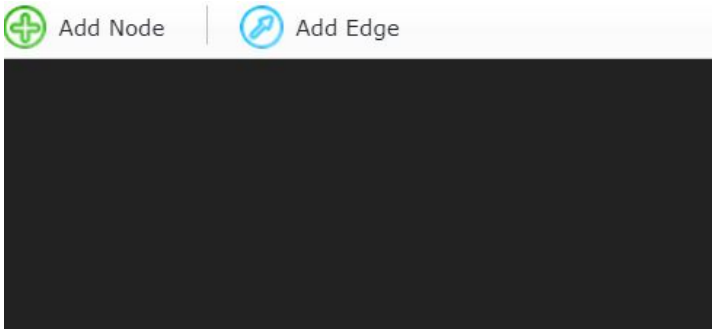
There are also some helping and printing functions.

Simple user guide & Sample runs

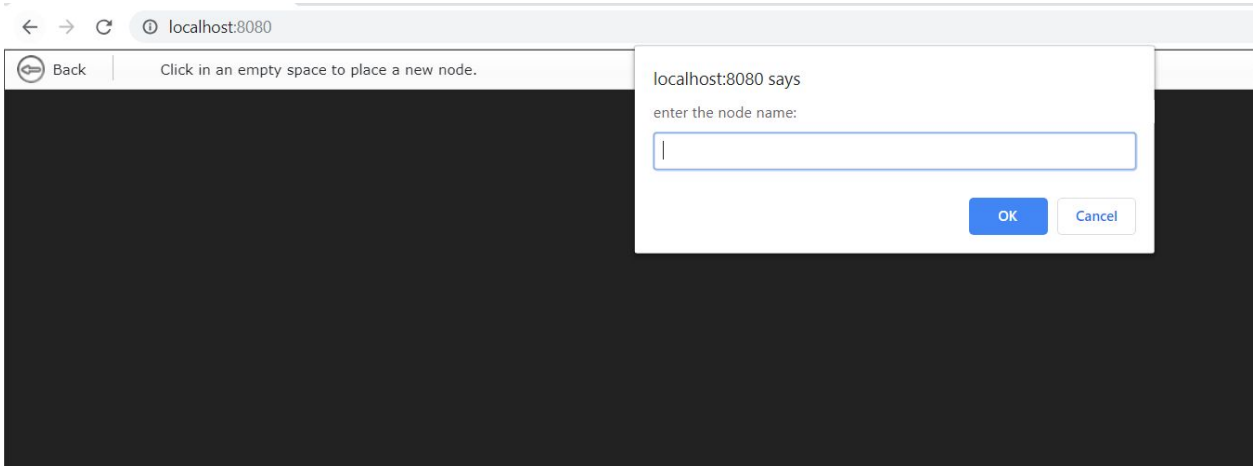
The screen on opening the page.



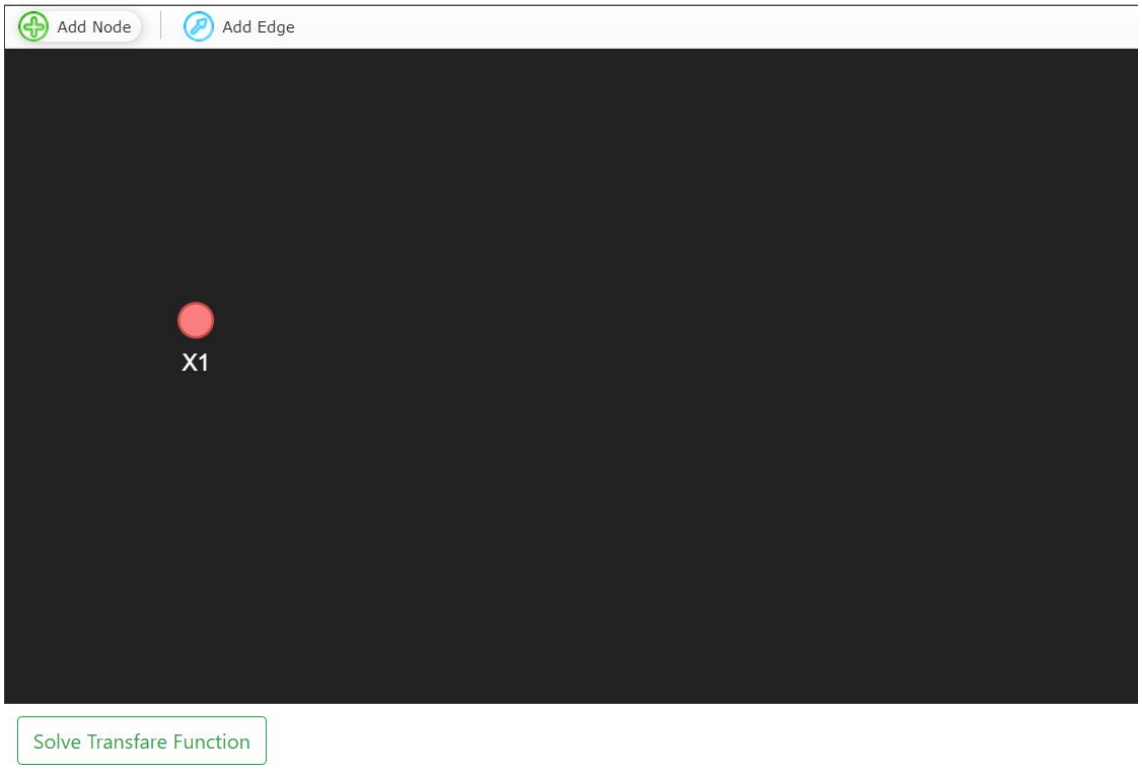
On pressing Edit it shows two options



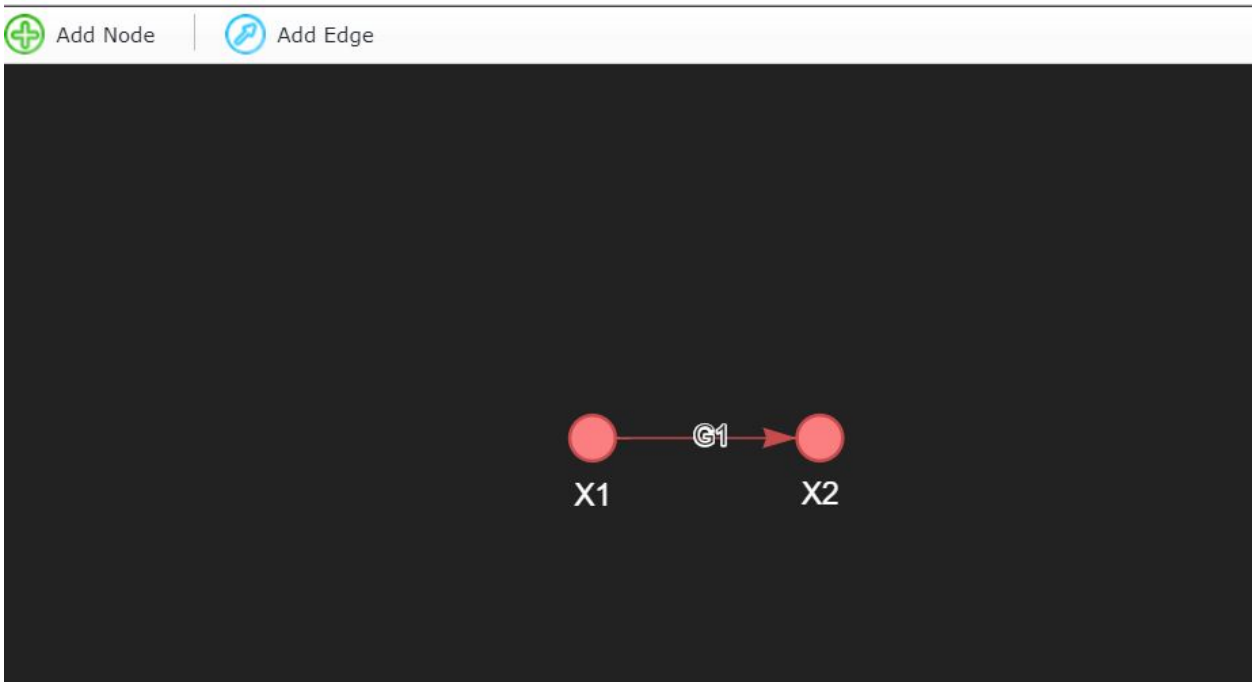
Choosing add node and hitting anywhere on the canvas a message pops up and requests a name to the node (label)

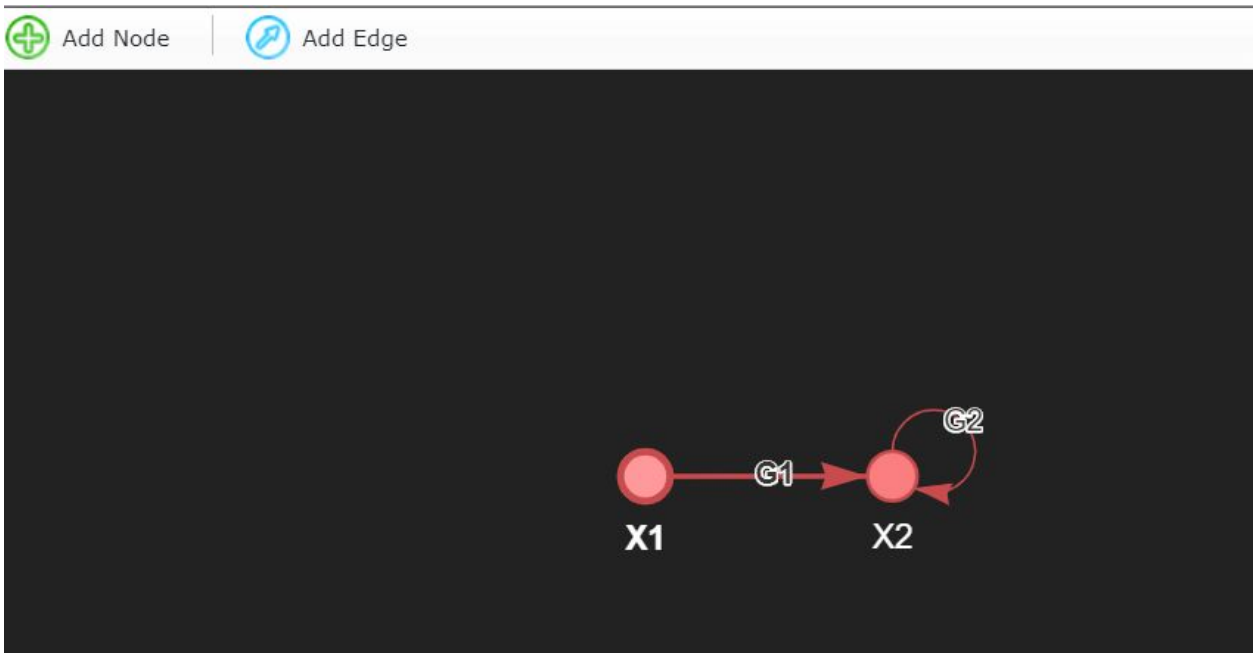


After getting the node label it draws a node in the canvas and pins it in its place if it's the first node

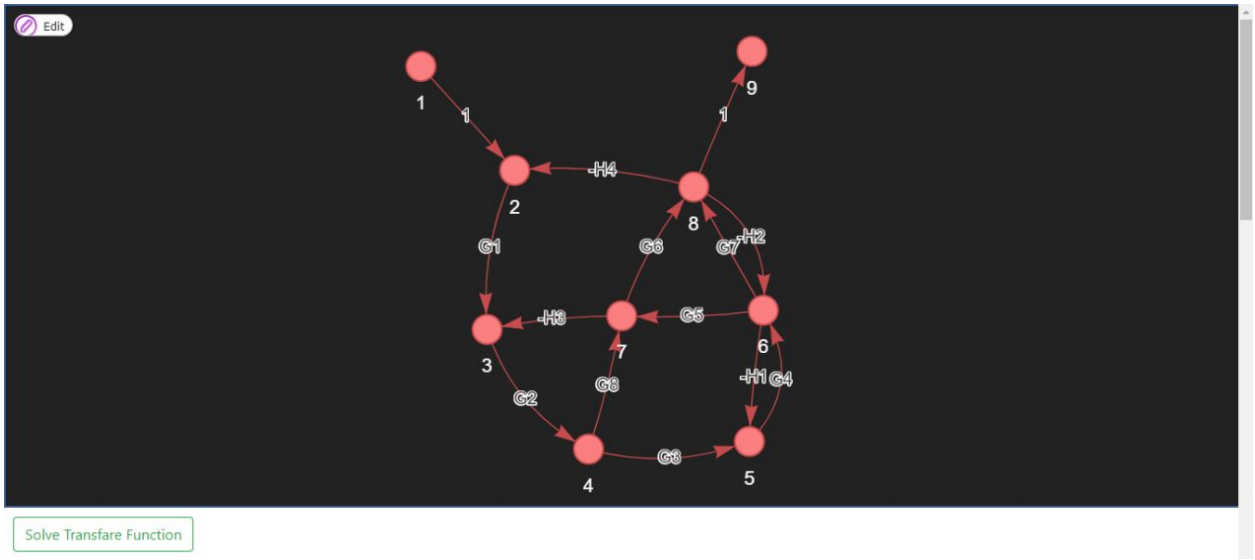


After Drawing another node/nodes we can choose to add an edge between them or an edge to any node itself simply by dragging an arrow from and to the required nodes/node the page also requests for a label for that edge.

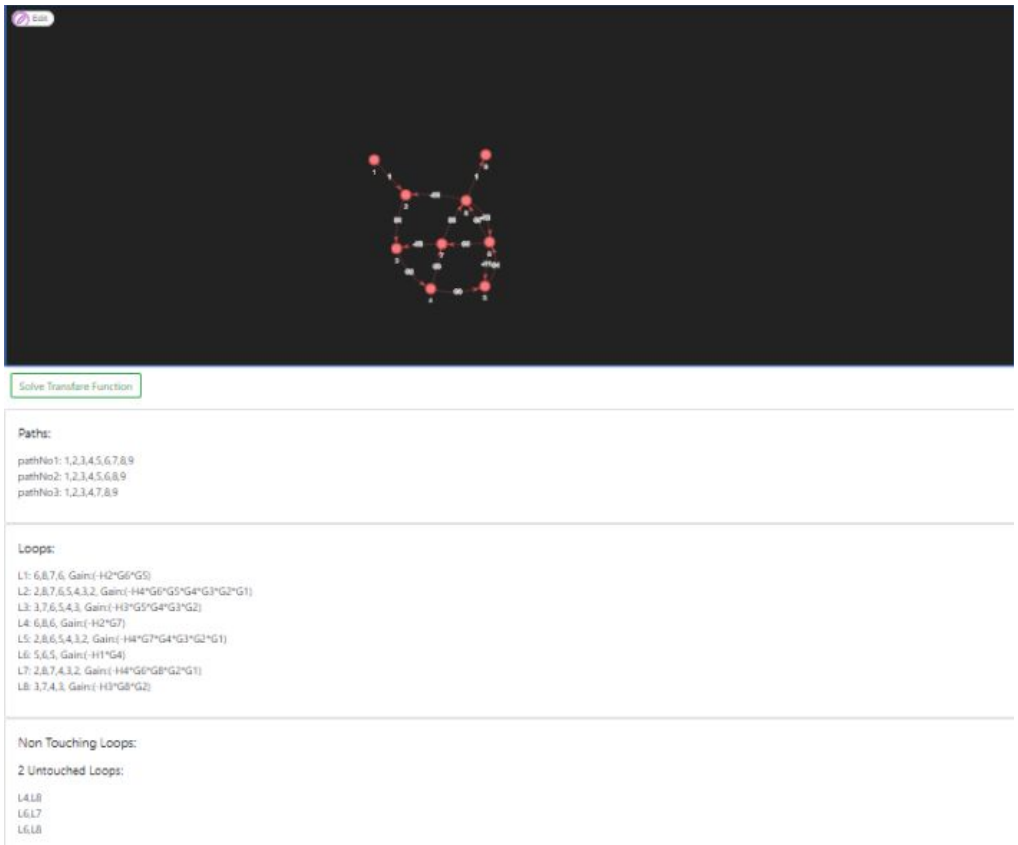




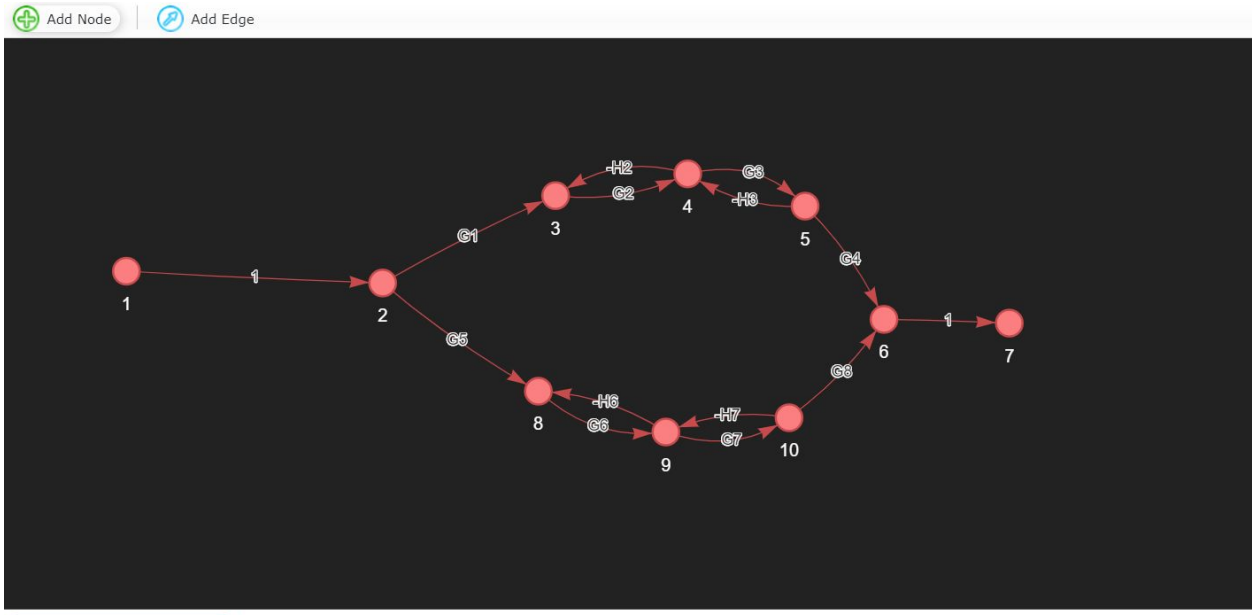
The user can now build his graph easily



Pressing the “Solve TF” Button it prints the required data to the HTML page:



Here are some clear sample runs



Solve Transfare Function

Paths:

pathNo1: 1,2,3,4,5,6,7
pathNo2: 1,2,8,9,10,6,7

Loops:

L1: 4,5,4, Gain:(-H3*G3)
L2: 3,4,3, Gain:(-H2*G2)
L3: 9,10,9, Gain:(-H7*G7)
L4: 8,9,8, Gain:(-H6*G6)

Non Touching Loops:

2 Untouched Loops:

L1,L3
L1,L4
L2,L3
L2,L4

Δ=

1 -((-H3*G3)+(-H2*G2)+(-H7*G7)+(-H6*G6))+((-H3*G3)*(-H7*G7)+(-H3*G3)*(-H6*G6)+(-H2*G2)*(-H7*G7)+(-H2*G2)*(-H6*G6))

2 Untouched Loops:

L1,L3
L1,L4
L2,L3
L2,L4

Δ=

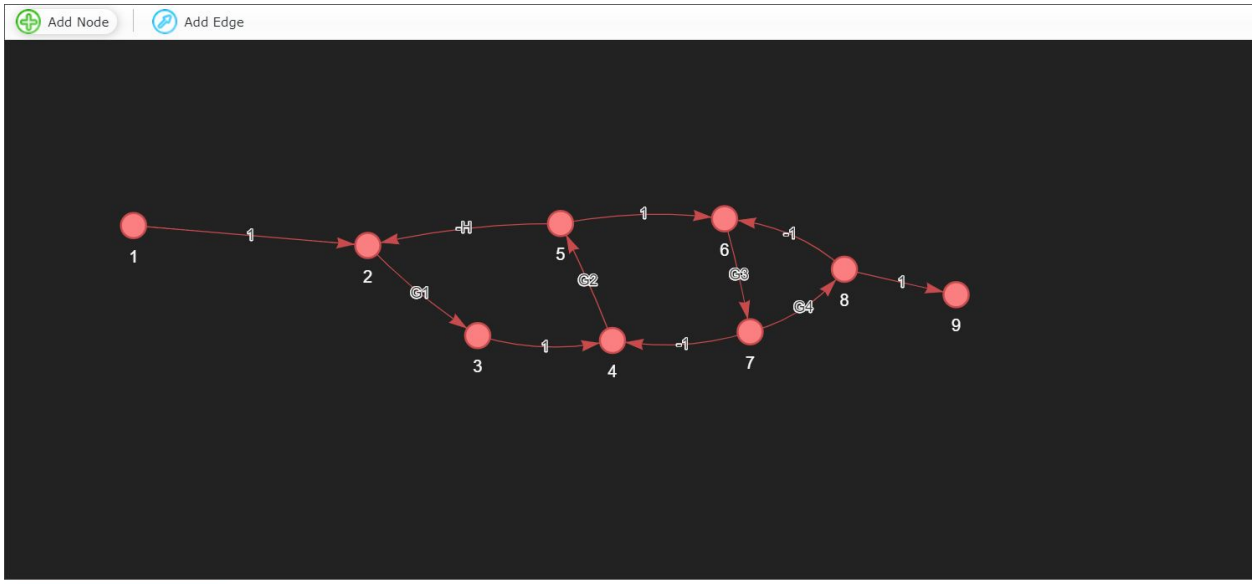
1 -((-H3*G3)+(-H2*G2)+(-H7*G7)+(-H6*G6))+((-H3*G3)*(-H7*G7)+(-H3*G3)*(-H6*G6)+(-H2*G2)*(-H7*G7)+(-H2*G2)*(-H6*G6))

Δ(i)=

Δ1= 1 -((-H7*G7)+(-H6*G6))
Δ2= 1 -((-H3*G3)+(-H2*G2))

Transfer Function=

(1*G1*G2*G3*G4*1)*(1 -((-H7*G7)+(-H6*G6)))+(1*G5*G6*G7*G8*1)*(1 -((-H3*G3)+(-H2*G2))) / 1 -((-H3*G3)+(-H2*G2)+(-H7*G7)+(-H6*G6))+((-H3*G3)*(-H7*G7)+(-H3*G3)*(-H6*G6)+(-H2*G2)*(-H7*G7)+(-H2*G2)*(-H6*G6))



Solve Transfare Function

Solve Transfare Function

Paths:

pathNo1: 1,2,3,4,5,6,7,8,9

Loops:

L1: 6,8,7,6, Gain:(-1*G4*G3)
L2: 4,7,6,5,4, Gain:(-1*G3*1*G2)
L3: 2,5,4,3,2, Gain:(-H*G2*1*G1)

Non Touching Loops:

2 Untouched Loops:

L1,L3

Δ=

1 -((-1*G4*G3)+(-1*G3*1*G2)+(-H*G2*1*G1))+((-1*G4*G3)*(-H*G2*1*G1))

Non Touching Loops:

2 Untouched Loops:

L1,L3

Δ=

1 -((-1*G4*G3)+(-1*G3*1*G2)+(-H*G2*1*G1))+((-1*G4*G3)*(-H*G2*1*G1))

Δ(i)=

Δ1= 1

Transfer Function=

(1*G1*1*G2*1*G3*G4*1)*(1) / 1 -((-1*G4*G3)+(-1*G3*1*G2)+(-H*G2*1*G1))+((-1*G4*G3)*(-H*G2*1*G1))

Repo and Project link

https://github.com/ahmedezeny/Signal-flow-graph-solver