Computer & Systems Engineering Department
Automatic Control Course
Second Year
Programming Assignment#1
Signal Flow Graph

BY
Salma Bakr (37)
Mennatallah Hany (71)
Nourane Ihab (75)
Due Date : 12/5/2015

Problem Statement:

Given : Signal flow graph representation of the system. Assume that total number of nodes and numeric branches gains are given.

Required:

1- Graphical interface.
2- Draw the signal flow graph showing nodes, branches ,gains, …
3- Listing all forward paths, individual loops, all combination of n non-touching loops.
4- The values of Delta , Delta 1, …,Delta m where m is number of forward paths.
5- Overall system transfer function.

- Java programming language is used to implement this program Draw nodes , branches ,loops, and gains of signal flow graph by drag and drop them in the drawing area. get all forward paths, individual loops, all combination of n non-touching loops. Compute the values of Delta , Delta 1, …,Delta n ,where n number of nodes . Compute overall system transfer function.

Main modules

GUI

- contains program graphical user interface .

Cycles

- get all loops of signal flow graph.

Generate forward paths

- get all forward paths

Compute(Calculate class)

- Compute the values of Delta , Delta 1, …,Delta n ,where n number of nodes .
- Compute overall system transfer function.

Cycles & Forward Paths

Description :

Every node has an integer number representing it (node a0 ,a1,a2,a3,...)

"Edge" is a class that contains :

1- integer "from" (Starting node number)
2- integer "to" (ending node number)
3- double "gain" (weight between the two nodes)

Module 1 :

(Path Generation)

Input from the GUI :

1- ArrayList of Edges representing all edges in gragh .
2 boolean 2D adjacency matrix .

DataStructures :

1- Path :: Array list of nodes(integers) *ex : a0a2a5 --> 0,2,5

2- All Paths :: Array list of Paths (Arraylist of integers)
*ex : a0a3a4 a1a5a8 --> (0,3,4),(1,5,8)

3- All Paths with weights :: Array list of Edges // final output

Algorithm :

Firstly, get all forward paths without weights

```
+SearchForwardPath ( Start , Terminal ) { //start and end of the whole graph
initially
                Path.add(start)
                for ( i=0 to total number of nodes ){
                        if( starting node connected to i-th node ) {
                                if( i-th node is terminal ) { //finished a path

                                        path.add ( terminal)
                                        AllPaths.add( path )
                                        path.removeLast

                                }else if ( i-th node already in path ) { // loop
                                        //do nothing
                                }else{
//recursion                             SearchForwardPath ( i-th node , Terminal )
//back tracking                       Path.removeLast
                                }
                        }
                }
        }
```

Now we have : Array list of Paths

Secondly , do weight Conversion (add weights to the paths)

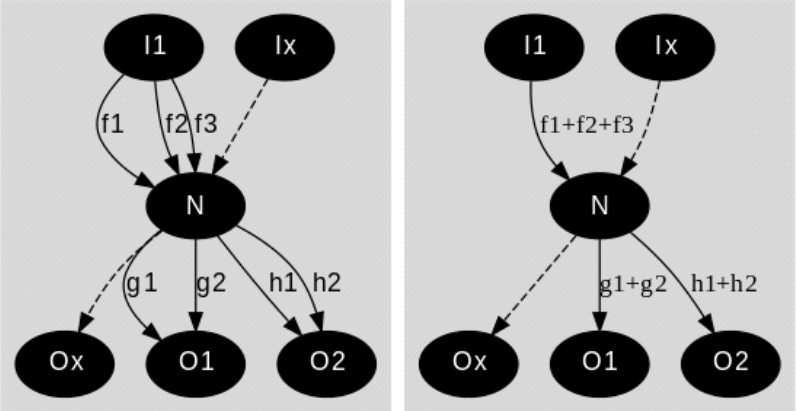
for each path take every two consecutive nodes (an edge) get it's weight .. then add path with weights to "All Paths with weights"

Assumption :

For Parallel edges (that has same start and same end but different weights)

it is considered one path and those edges' gains are added

Parallel edges. Replace parallel edges with a single edge having a gain equal to the sum of original gains.



source :wikipedia

==========

Module 2 :

Cycles

Input from the GUI :

1- - ArrayList of Edges representing all edges in gragh .
2- boolean 2D adjacency matrix .

DataStructures :

1- Cycle :: Array list of nodes(integers)
*ex : a0a2a5a0 --> 0,2,5,0
- All Cycles :: Array list of Paths (Arraylist of integers)
*ex : a0a3a4a0 , a1a5a8a1 --> (0,3,4,0),(1,5,8,1)
3- All Cycles with weights :: Array list of Edges
// final output

Algorithm :

Firstly, get all cycles

```
for(itr= 0 : total number of nodes){
        searchLoops(itr, itr);
}
 searchLoops( Originalstart,  RecursiveStart) {
        Path.add(RecursiveStart);
        for ( i=0 to total number of nodes ){
                //check from adjacency matrix
                if ( RecursiveStart connected to i-th node ) {
                        if (i is Originalstart) {
                                Path.add(Originalstart)
                                AllPaths.add( path )
                                path.removeLast
                        } else if  ( i-th node already in path ) { // loop
                                //do nothing
                        } else {
                                if (i >= Originalstart) {
//recursion                                       searchLoops(Originalstart, i-th node );
//back tracking                                   Path.removeLast
                                }
                        }
                }
        }
}
```

Now we have : Array list of all cycles

Secondly , do weight Conversion (add weights to the cycles)

for each cycle take every two consecutive nodes (an edge) get it's weight .. then add cycle with weights to "All cycles with weights"

==========

Module 4:

Take objects of forward paths and cycles as input

Procedures

- First calculate forward paths gains
- Second calculate gains of loops
- Create bit mask for each forward path
- Create bit mask for each loop
- Delta function to calculate deltas' of forward path and calculate the general delta using complete search algorithms to find all untouched loops

Algorithm used at this class

Input : array list of array list of edges of forward paths
: array list of array list of edges of loop

For loop on each forward path
Calculate gain of the path and store it in array list of double
Calculate bit mask of the path and store it in array list of integer
For loop on each loop path
Calculate gain of the loop and store it in array list of double
Calculate bit mask of the loop and store it in array list of integer

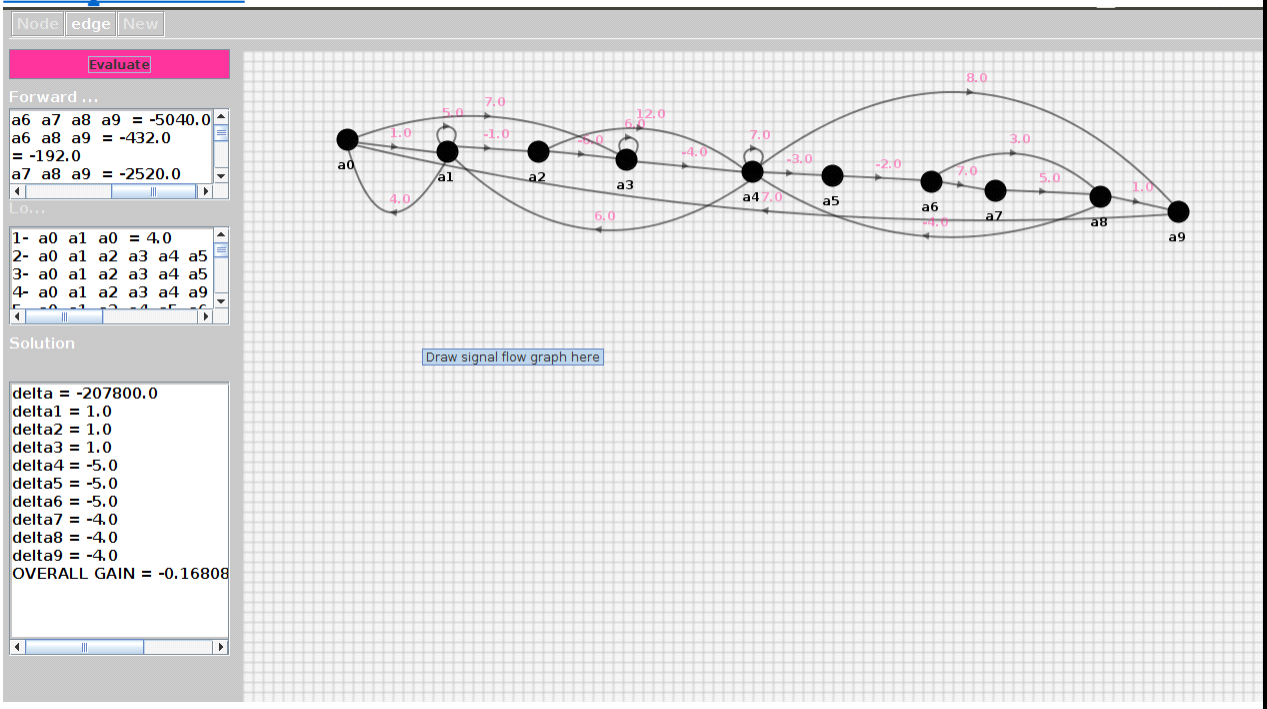
Then loop on all combination of cycles
To find all untouched loops
Return strings of forward paths with gains and loops with gains and deltas

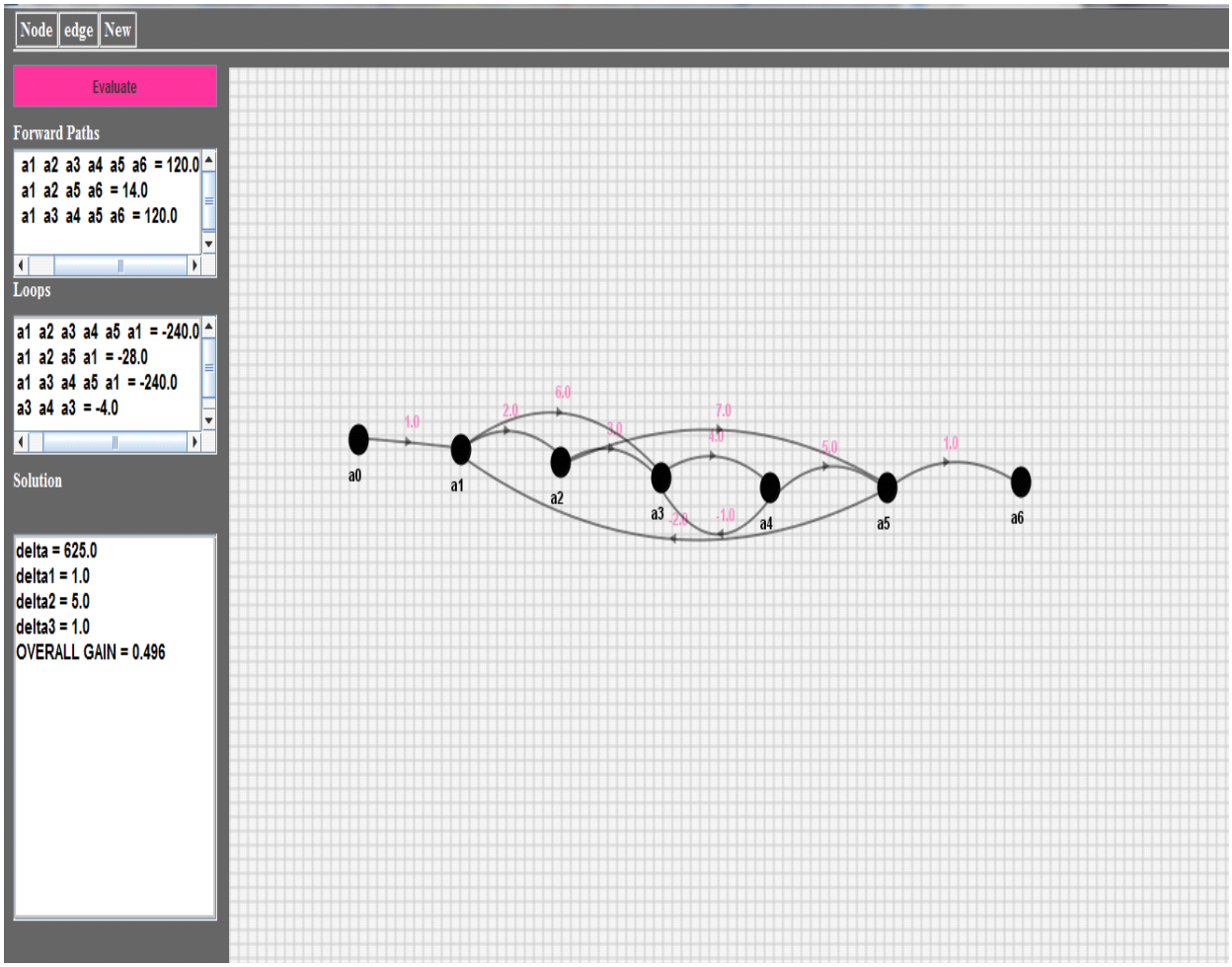
End algorithm

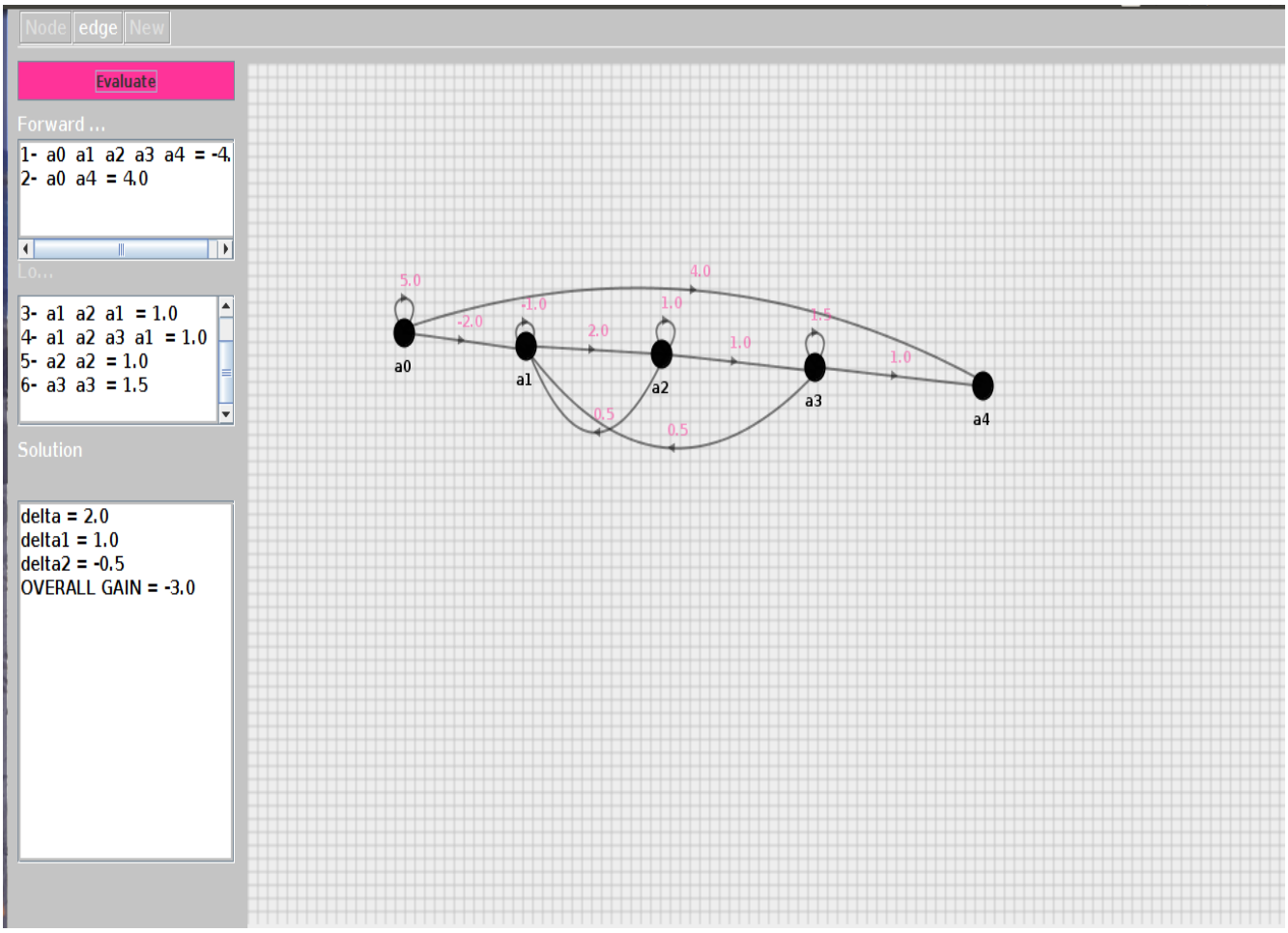
Data structure used
Array list of integers doubles , array list of array list of edges

For loop on each forward path
Calculate gain of the path and store it in array list of integers
Calculate bit mask of the path and store it in array list of integer

Sample runs


Node
edge
New
Evaluate
Forward ...
a6 a7 a8 a9 = -5040.0
a6 a8 a9 = -432.0
= -192.0
a7 a8 a9 = -2520.0
Lo...
1- a0 a1 a0 = 4.0
2- a0 a1 a2 a3 a4 a5
3- a0 a1 a2 a3 a4 a5
4- a0 a1 a2 a3 a4 a9
Solution
delta = -207800.0
delta1 = 1.0
delta2 = 1.0
delta3 = 1.0
delta4 = -5.0
delta5 = -5.0
delta6 = -5.0
delta7 = -4.0
delta8 = -4.0
delta9 = -4.0
OVERALL GAIN = -0.16808
a0
a1
a2
a3
a4
a5
a6
a7
a8
a9
1.0
5.0
7.0
4.0
-1.0
-5.0
12.0
6.0
-4.0
7.0
-3.0
-2.0
8.0
7.0
3.0
5.0
1.0
-4.0
6.0
Draw signal flow graph here


Node
edge
New
Evaluate
Forward Paths
a1 a2 a3 a4 a5 a6 = 120.0
a1 a2 a5 a6 = 14.0
a1 a3 a4 a5 a6 = 120.0
Loops
a1 a2 a3 a4 a5 a1 = -240.0
a1 a2 a5 a1 = -28.0
a1 a3 a4 a5 a1 = -240.0
a3 a4 a3 = -4.0
Solution
delta = 625.0
delta1 = 1.0
delta2 = 5.0
delta3 = 1.0
OVERALL GAIN = 0.496
a0
a1
a2
a3
a4
a5
a6
1.0
2.0
6.0
-1.0
7.0
4.0
5.0
1.0
-2.0
-1.0



User guide

- to draw nodes select node button and click with mouse in the drawing area where you want to draw graph.
- To draw branch select branches button and click with mouse in the starting node and continue pressing on mouse until end node.
- To evaluate the values of Delta , Delta 1, ...,Delta m where m is number of forward paths and overall system
- New button to create new drawing area .

Assumptions ::

- starting node is the first to be drawn .
- sink node is the last to be drawn .
- forward paths are named with nodes' names
- cycles are named with nodes' names
- parallel edges are not considered as different paths