Signal Flow Graph

12 / 5 / 2015

Signal Glow Graph is a systematic method to solve the overall transfer function of any systems type , the project allows you to draw your SFG and solves it with many options , Project is implemented in Java with OOP Design patterns

Team members

Ahmed Khaled
ID : 9

Khaled Mohamed El-Tahan
ID : 26

Contents

Problem Statement

Given

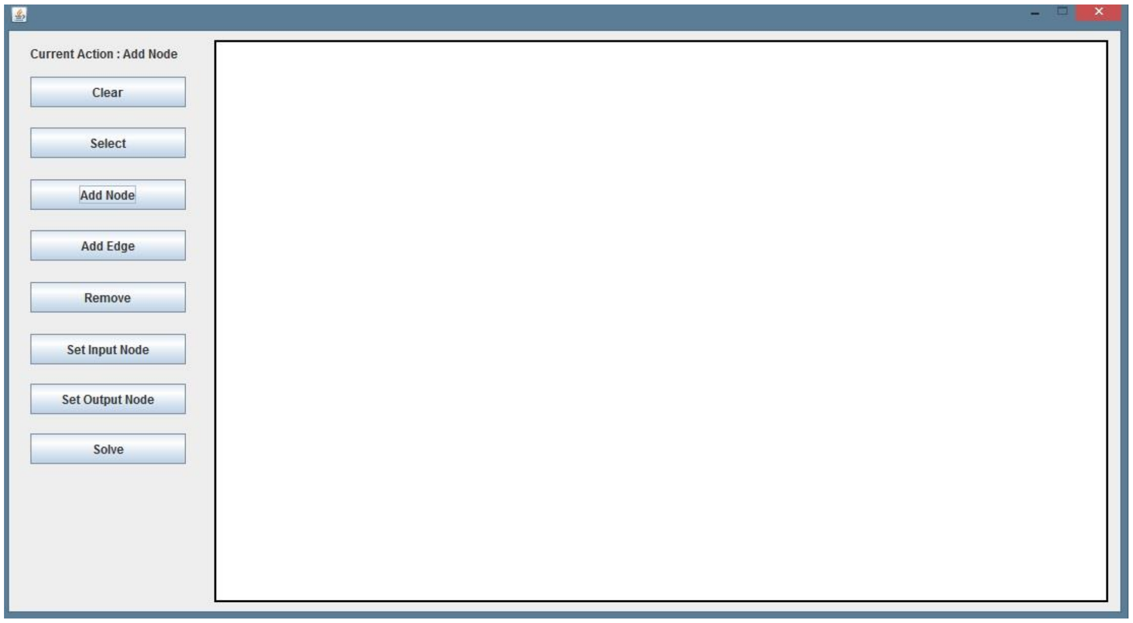
Signal flow graph representation of the system. Assume that total number of nodes and numeric branches gains are given.

Requirements

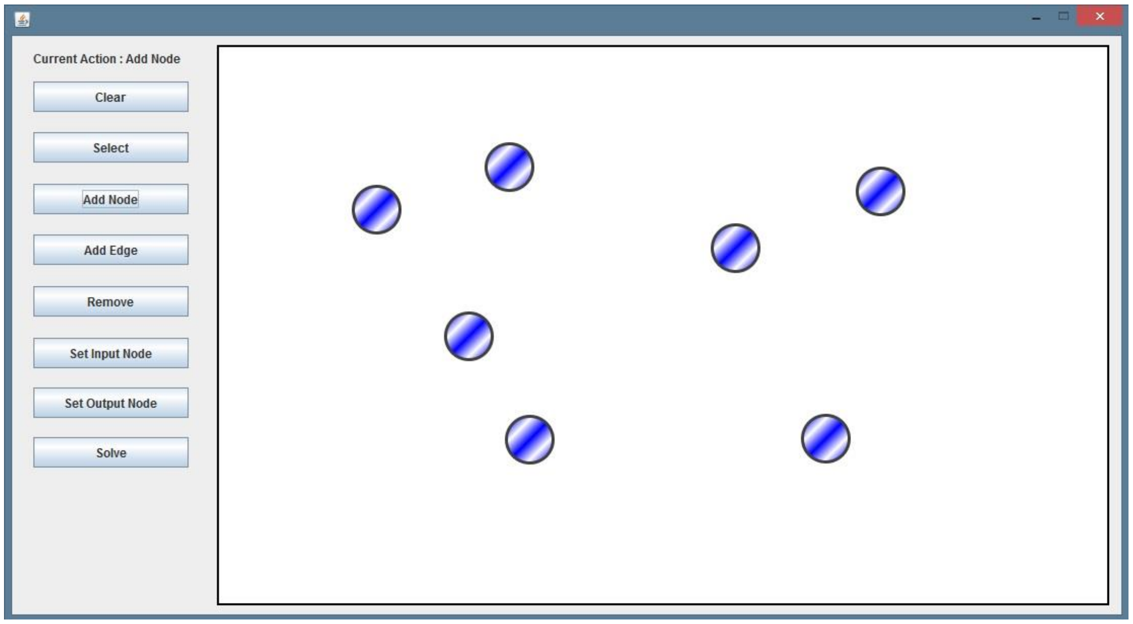
- Graphical Interface
- Draw The Signal Flow Graph Showing nodes , branches & gains
- Listing all forward paths , individual loops & all combination of n non-touching loops
- The values of Δ,Δ1, Δ2,…. Δm where m is the number of forward paths
- Over all system transfer function

Main Features

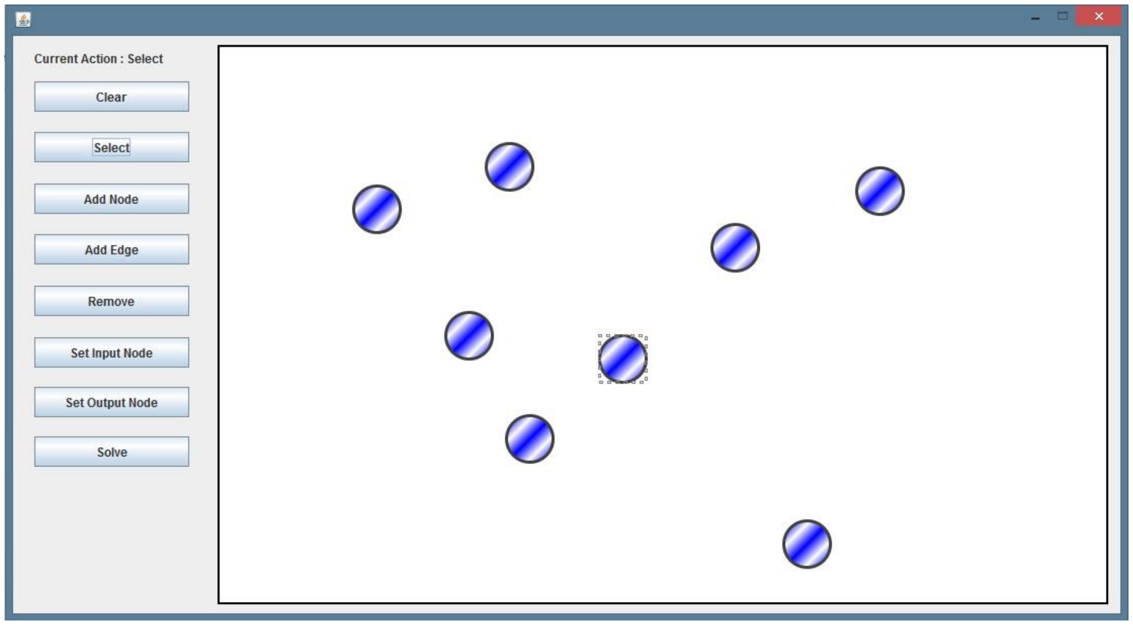
- Select Your Favorite Mode , Current Action will tell you the running mode



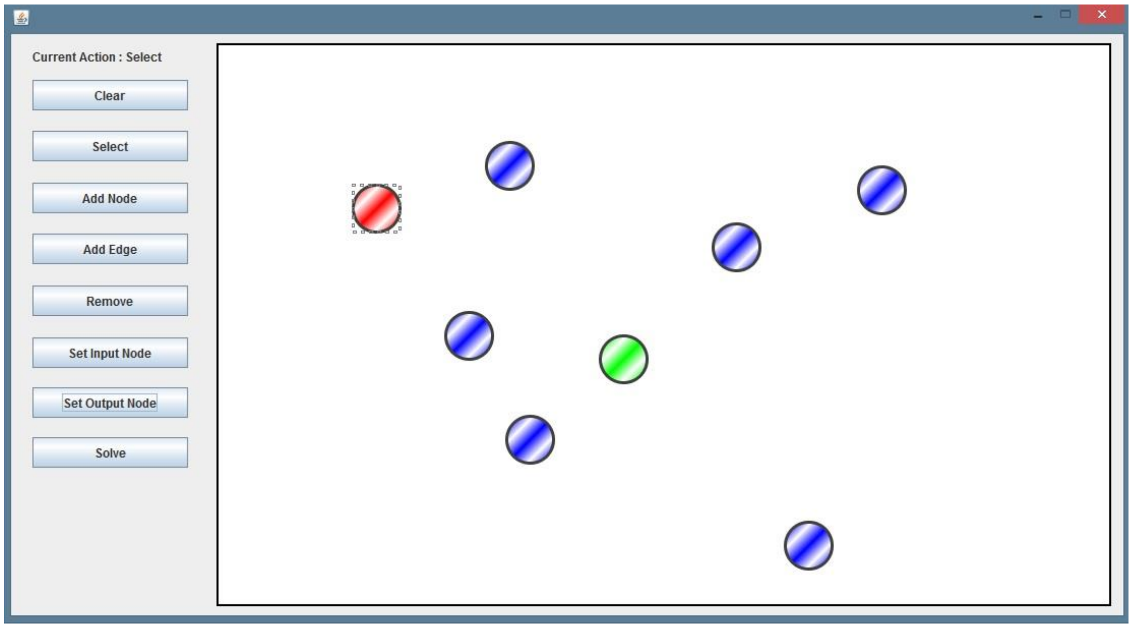
- Add nodes , Simple touches of beauty are added



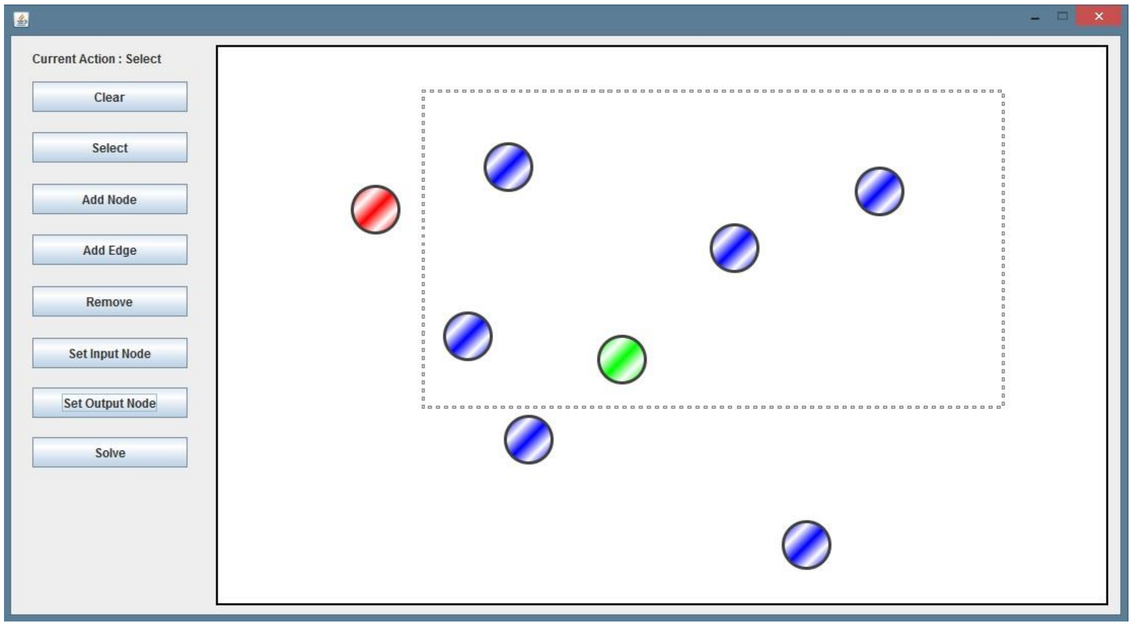
- Select any node , Move it or remove it , A dotted frame is showed for the selected node



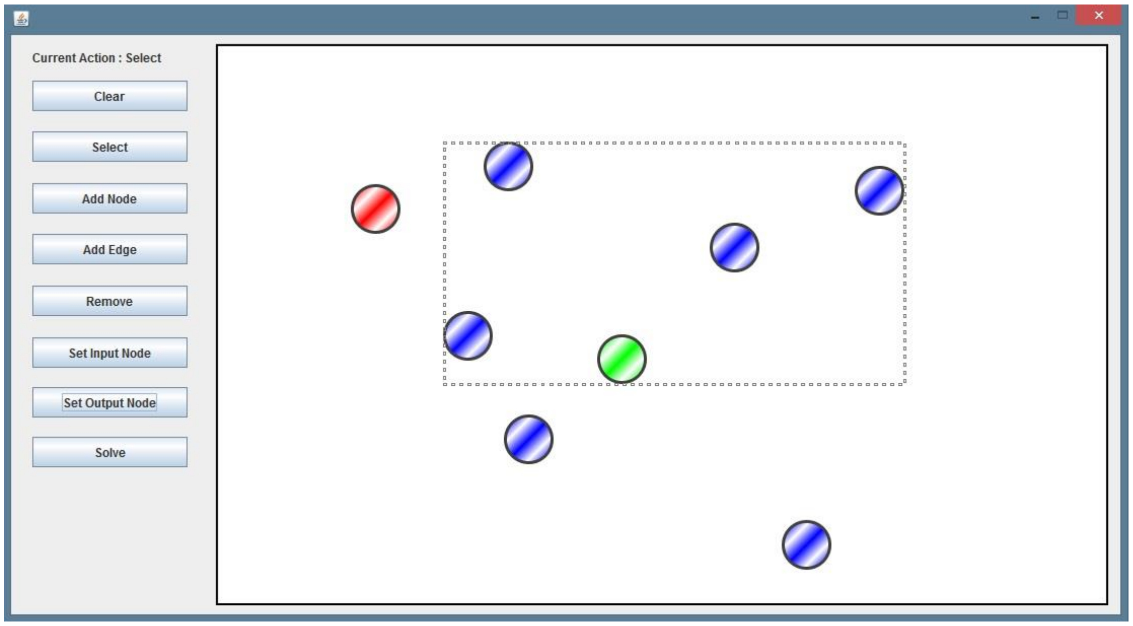
- You can denote Selected Node as input/output and change it at any time , Green for input , Red for output



- You can select set of nodes , by pressing the mouse and dragging , just the same as windows selecting

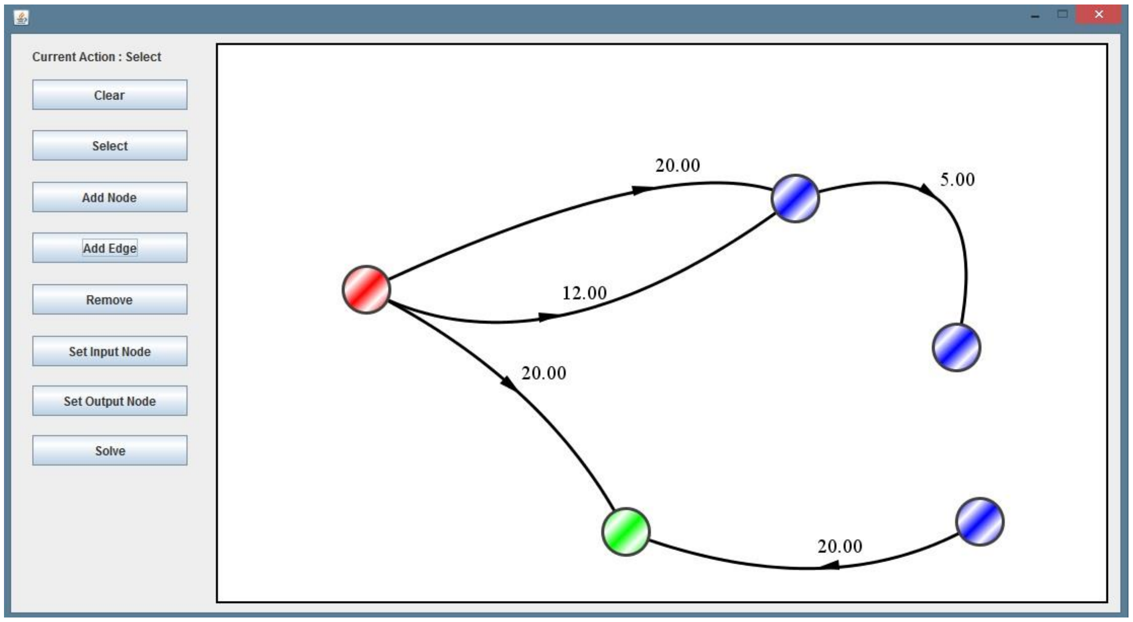


- Nodes after selected , Nodes can be removed at once

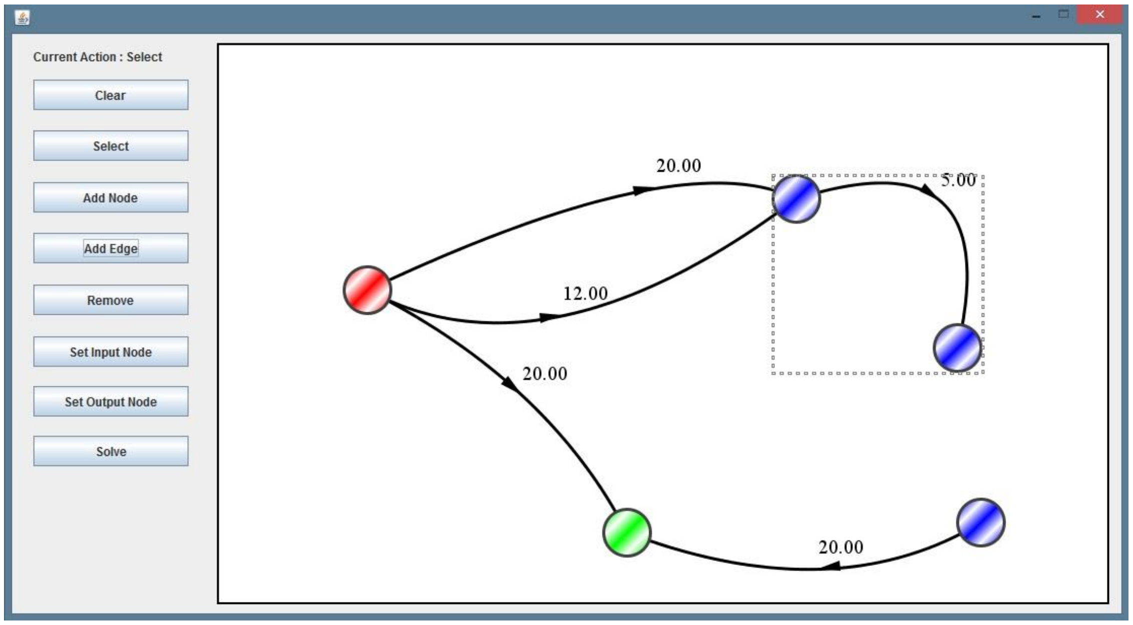


Main Features

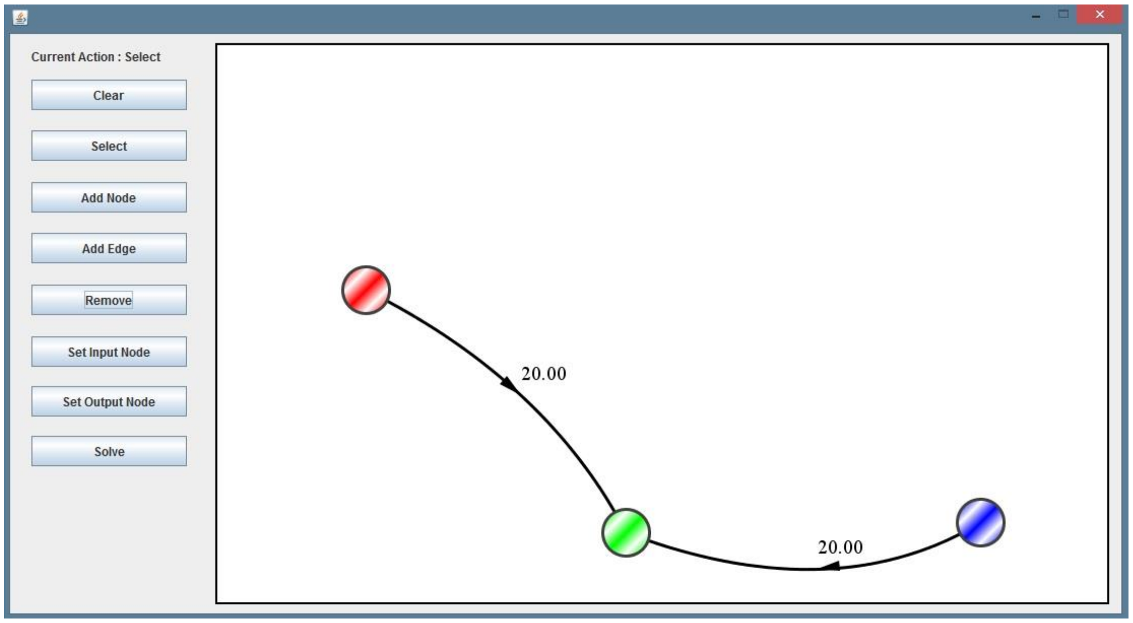
- Adding Edges , You can move the nodes to have a better looking , clearer edges



- Selecting Nodes



- After Removing the selected nodes



Data Structure

- Edge class : is used to represent a single edge with three attributes :-
    - [from , to , the weight of the edge]
- ArrayList of edges : is used to save a single forward path as a group of consecutive edges from the source node to the sink node or to represent a single loop as a group of consecutive edges from a node back to itself .
- ArrayList of the previous structure : is used to save all the forward paths and another one to save all the loops in the graph.
- VirtualNode : is used to store the required information to draw a node such as :-
    - [the coordinates of its center , the radius , its color , a frame surrounding it]
- VirtualEdge : is used to store the required information to draw the edge :-
    - [the two virtual nodes it connects between , some points to draw a direct arrow , label represent the gain of the edge].

Main Modules

The Main Module follows the MVC Object Oriented Design Patterns , which separates the whole project into three main modules

1. Model :-
   - Edge : contains the required information to represent a single edge edge(from , to , weight)
   - Core : contains the methods to generate forward paths , loops , all combination of n non-touching loops, deltas and the transfer function .
2. View :-
   - the view gives the user the ability to add nodes , edges , remove singe or group of nodes or edges , move single node , specify the input and output node and show the result details.
3. Control :-
   - the control is responsible for constructing a graph from the virtual drawn graph from the view , using the core to solve the graph and sending the result details to the view to show it to the user

For more about the MVC design pattern please check the following link :-
http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller

Algorithms Used

Calculating The Transfer Function

```
double calcTF(int s , int t)
{
                computeCycles();
                generateForwardPaths(s, t);

                double TF = 0 ;
                delta[0] = calcDelta(-1);

                for (int I = 1; I < delta.length; i++) {

                        delta[i] = calcDelta(i-1);
                        TF += delta[i]*pathGain(i-1);
                }

                return TF/delta[0];
}
```

Algorithm Description :-

- Generate all loops in the graph .
- Generate all forward paths in the graph from the source node s to the sink node t .
- Calculate the delta of the denominator .
- Calculate all deltas after removing one forward path in each time .
- Use the deltas and gains of the forward paths to compute the Transfer Function .

Generate All Forward Paths

```
findPath(int I ,int target , Deque<Integer> q)
{
        if(i==target)
        {
                savePath(q);
                return ;
        }

        for (int j = 0; j < adjList[i].size(); j++) {

                int to = adjList[i].get(j).getTo();
                if(!visited[to])
                {
                        q.add(to);
                        visited[to] =true ;
                        findPath(to,target,q);
                        visited[to] =false ;
                        q.removeLast();
                }
        }

}
```

Algorithm Description :-

- Depth-first search with backtracking as a complete search technique is used to find all forward paths from the source node to the sink node.

Generate All Loops

```
dfs( int node , Stack<Integer> s ){
        visited[node] = true ;
        s.push(node);

        for(int i=0 ; i<adjList[node].size() ; ++i){
                int to = adjList[node].get(i).getTo() ;
                if( visited[to] ) // if there is a backward edge
          {
                        generateCycle(to,s);
                }
                else{
                        dfs( to , s );
                }
        }

        s.pop();
        visited[node] = false ;
}
```

Algorithm Description :-

- Depth-first search with backtracking is used to generate a spanning tree with back-edges indicate to the cycles in the graph and in case of specifying a cycle it would be saved in a data structure mentioned later.

Find All Combination of n non-touching loops

```
double loopsCombinsGain(int index ,int p ,    orward[] outCycles , int[] takenCycles)
{

        if(index==takenCycles.length){
                saveCombination(takenCycles);
                return getGain(takenCycles);
        }

        double sum  = 0 ;
        for (int I = p ; I < cycles.size(); i++) {

                if(!outCycles[i] && isNonTouching(I,takenCycles,index))
                {
                        takenCycles[index] = I;
                        sum += loopsCombinsGain(index+1,i+1,outCycles,takenCycles);
                }

        }

        return sum ;
}
```

Algorithm Description :-

- Using complete search to find all combination of n non-touching loops and to calculate the sum of the gains of all combination for a specific n .

Calculating Deltas

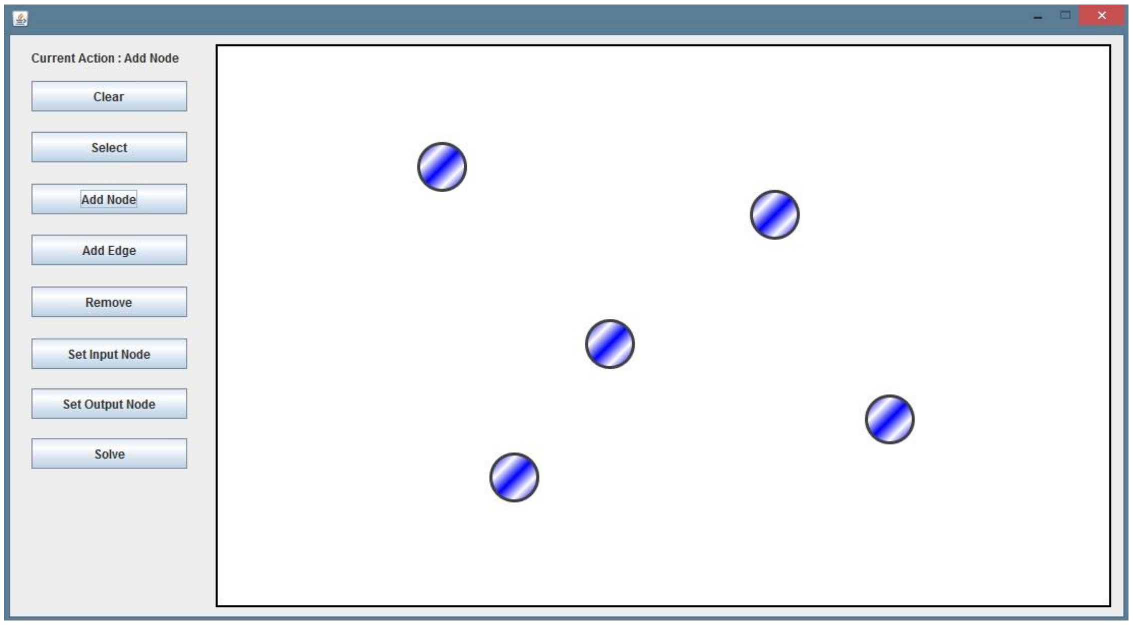
```
double calcDelta(int outPath)
{
        double delta = 1  , gain=0 ;
        int cnt = 1 ;
          orward[] outCycles = new    orward[cycles.size()];
        int[] takenCycles ;

        if(outPath!=-1) // -1 to calculate the delta of the denominator
                outCycles = outCycles(outPath);  // get the cycles after removing the    orward
        do
        {
                takenCycles = new int [cnt];
                gain = loopsCombinsGain(0,0,outCycles,takenCycles);
                delta += Math.pow(-1, cnt)*gain;
                cnt++;

        }while(gain != 0) ;

        return delta ;
}
```

Algorithm Description :-

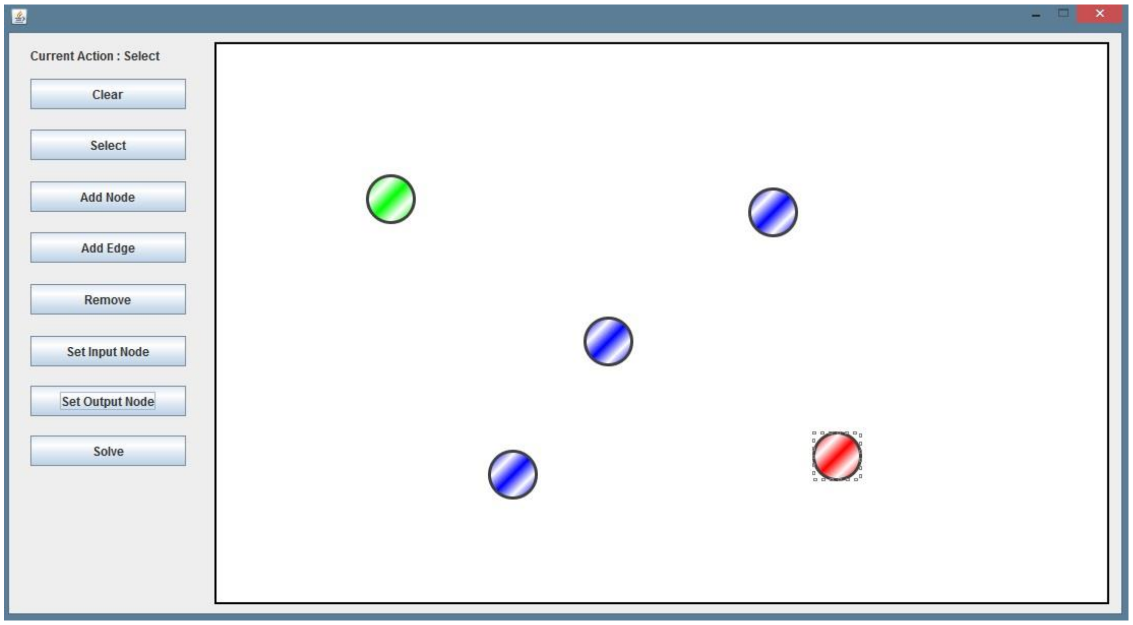
- Get the cycles after removing the path I to calculate Delta I .
- Calculate the sum of the gains of all combination of n non-touching loops and add the sum to the delta with its right sign .
- Repeat the previous step with increasing n , until there is no combination of n non-touching loops .
- return the calculated delta

User Guide

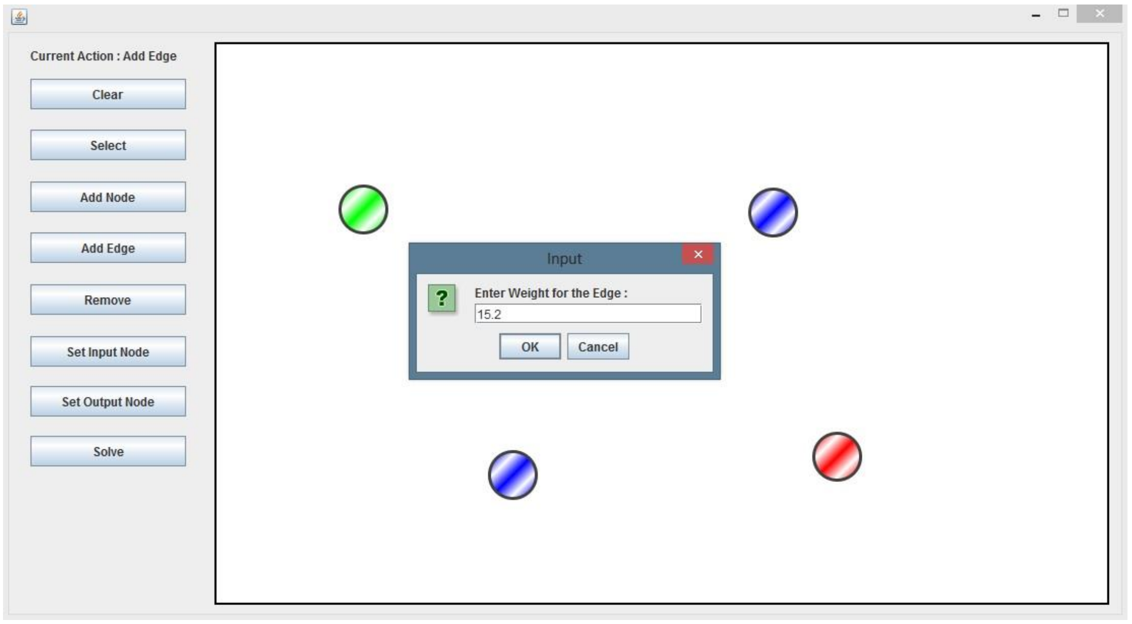
- Default Current Action is Select , is used to select any node and set it as input/output or move or remove it , is used also to select many nodes at the same time
- Go to Add nodes action , insert as many nodes as you love



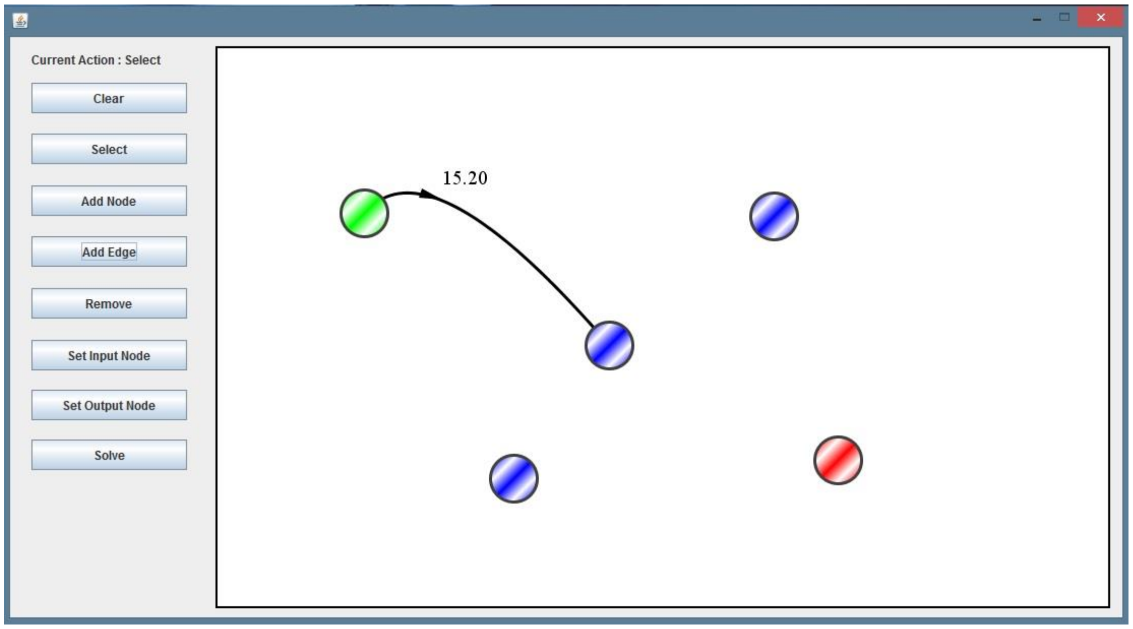
- Go to select mode , select your favorite node , set input ”green” and select other node as output ”red” , note : “The program wouldn’t solve without identifying input/output”



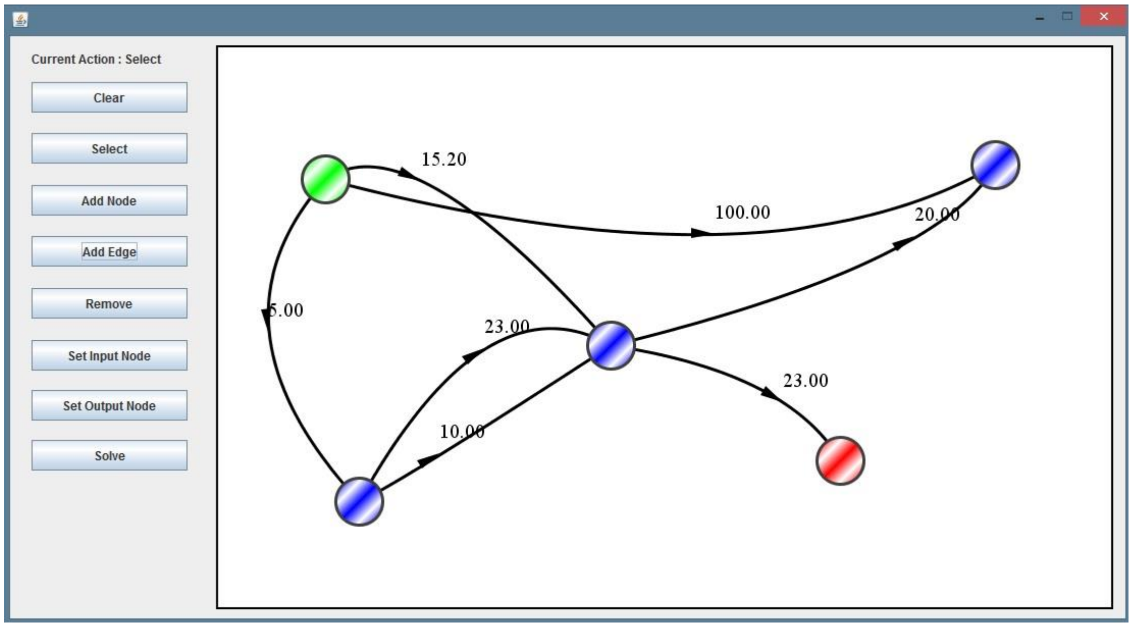
- Go to add edge , please always note the current action if you’ve done any problem , go to add edge then click on 1st node as source and 2nd node as target , you will asked for the gain as follows



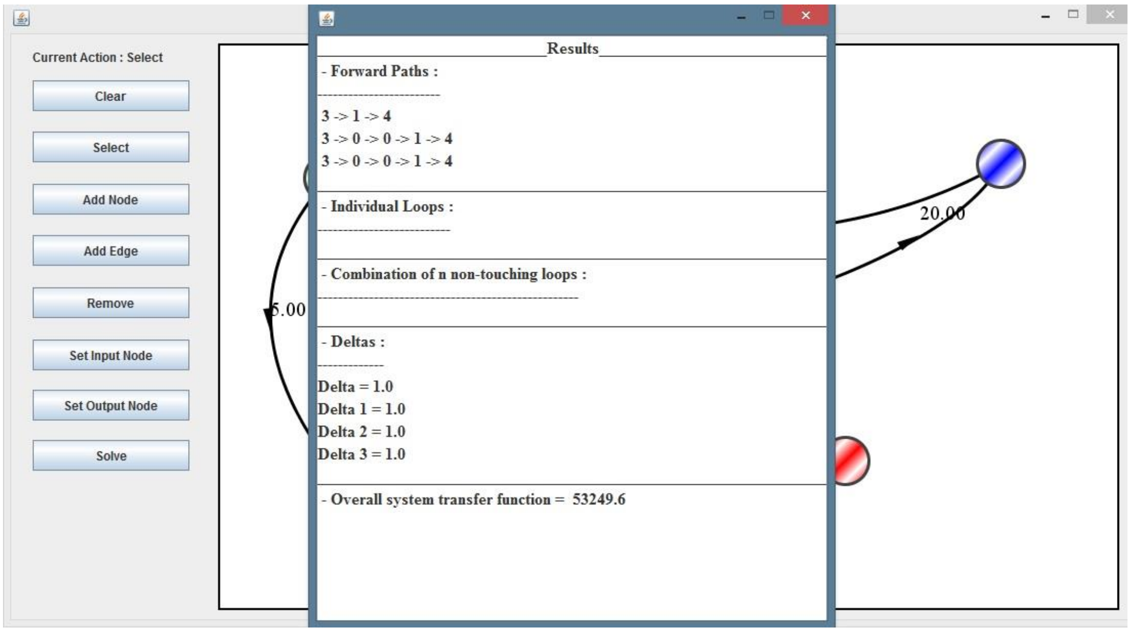
- After Adding the edge



- After Adding Many Edges



- Now Solve



Results

- Forward Paths :

3 -> 1 -> 4
3 -> 0 -> 0 -> 1 -> 4
3 -> 0 -> 0 -> 1 -> 4

- Individual Loops :

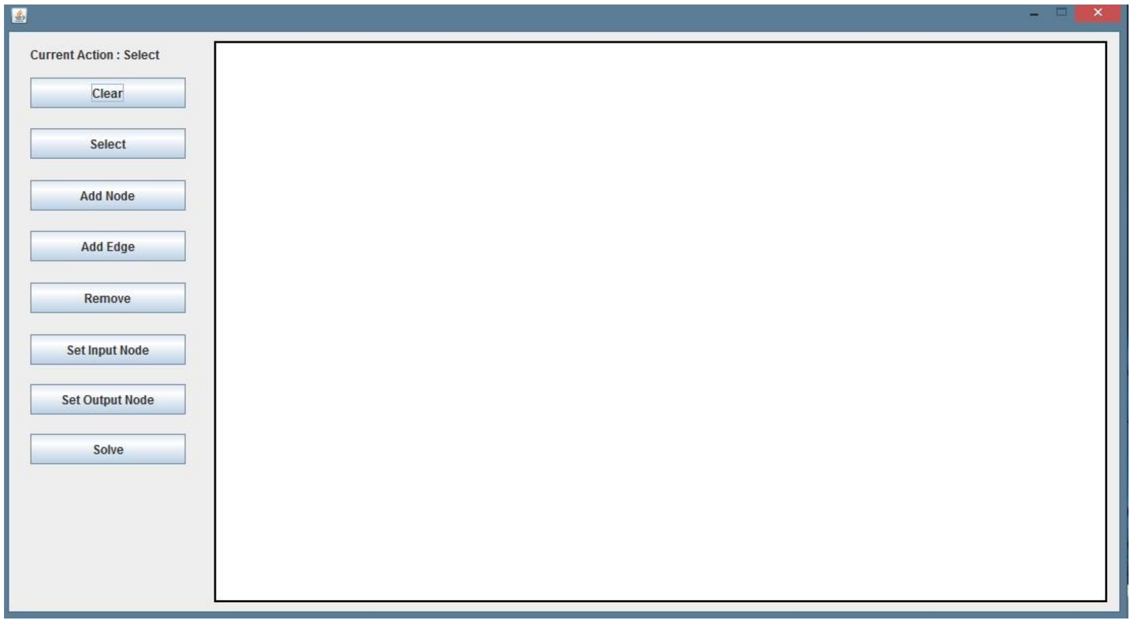
- Combination of n non-touching loops :

- Deltas :

Delta = 1.0
Delta 1 = 1.0
Delta 2 = 1.0
Delta 3 = 1.0

- Overall system transfer function = 53249.6

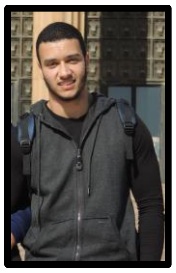
- Now Click on Clear



Sample Runs

I just Couldn’t resist , Thank you :D

Contact Information



Ahmed Khaled
eng.ahmed.khaled@hotmail.com
ID : 9

Khaled El-Tahan
Classic-life@live.com
ID : 26