CalPoly Pomona

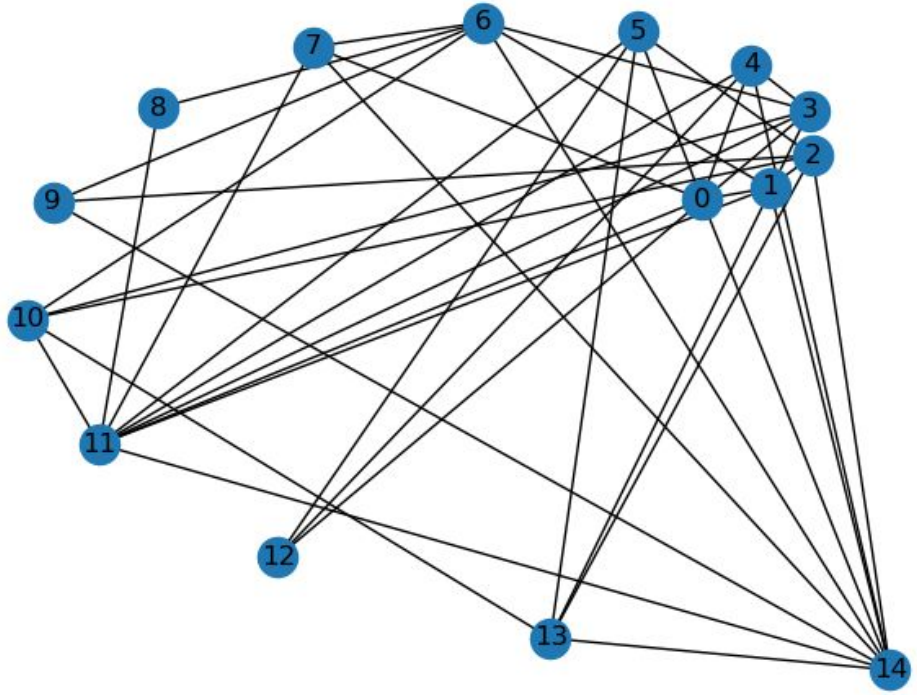
Group Assignment - 1

TEAM 5: Mohamed Abdul, Srijit Bhattacharya, Vikram Ramesh, and Gokul Srinath

Introduction

- Example Graph
- Graph Details - Original Network
- Abstract Implementation Details
- Pseudo-Code - Watts-Strogatz
- Watts-Strogatz Graph
- Graph Details - Watts-Strogatz
- Pseudo-Code - Barabasi-Albert
- Barabasi-Albert Graph
- Graph Details - Barabasi-Albert
- Data Set Graph Details - Experimental Results
- Summary
- Task Performed By Each Member

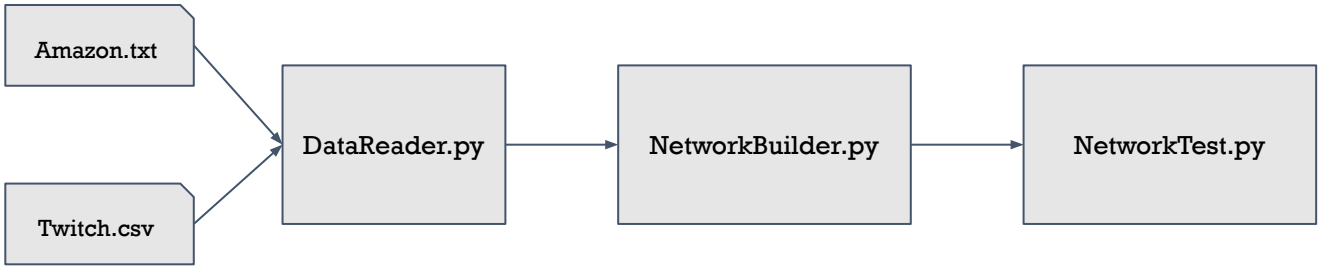
Example Graph



Graph Details - Original Network

Size	Average Degree	Average Clustering	Average Path
40	5.3	0.4	1.7

Abstract Implementation Details

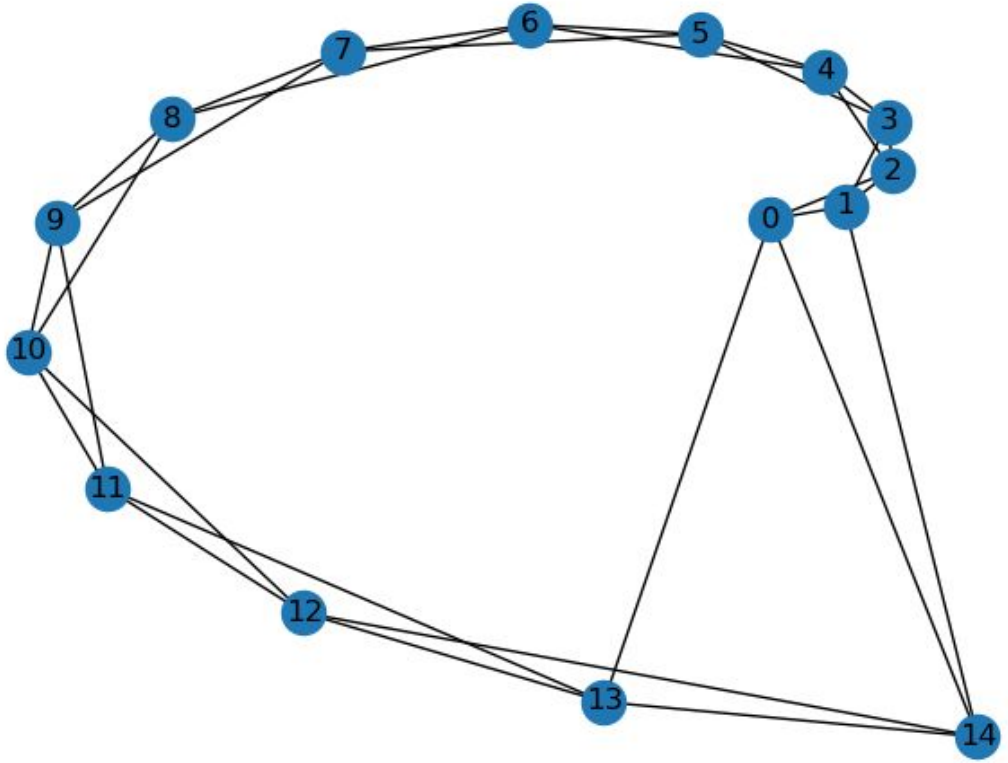



```
Main Methods
-------------------------------------------------------------------------------
def generateBarabasiAlbertNetwork(self, vertices:list, K:int) -> Graph
def generateWattsStrogatzNetwork(self, graph: Graph, K: int, beta: float) -> Graph
```

Watts-Strogatz - Pseudo Code

```
function generateWattsStrogatzNetwork(graph, K, beta):

    latticeGraph = generateRegularRingLatticeGraph(graph, K)
    nodes = getAllNodesFromGraph(latticeGraph)
    totalRewires = int(latticeGraph.size() * beta)

    for _ in range(totalRewires):
        sourceNode = getRandomNode(nodes)
        edges = getEdgesOfNode(latticeGraph, sourceNode)
        edgeToRemove = getRandomEdge(edges)
        latticeGraph.removeEdge(sourceNode, edgeToRemove)

        targetNode = getRandomNode(nodes)
        while latticeGraph.hasEdge(sourceNode, targetNode) or sourceNode == targetNode:
            targetNode = getRandomNode(nodes)

        latticeGraph.addEdge(sourceNode, targetNode)
    return latticeGraph
```

Watts-Strogatz - Graph



Graph Details - Watts-Strogatz

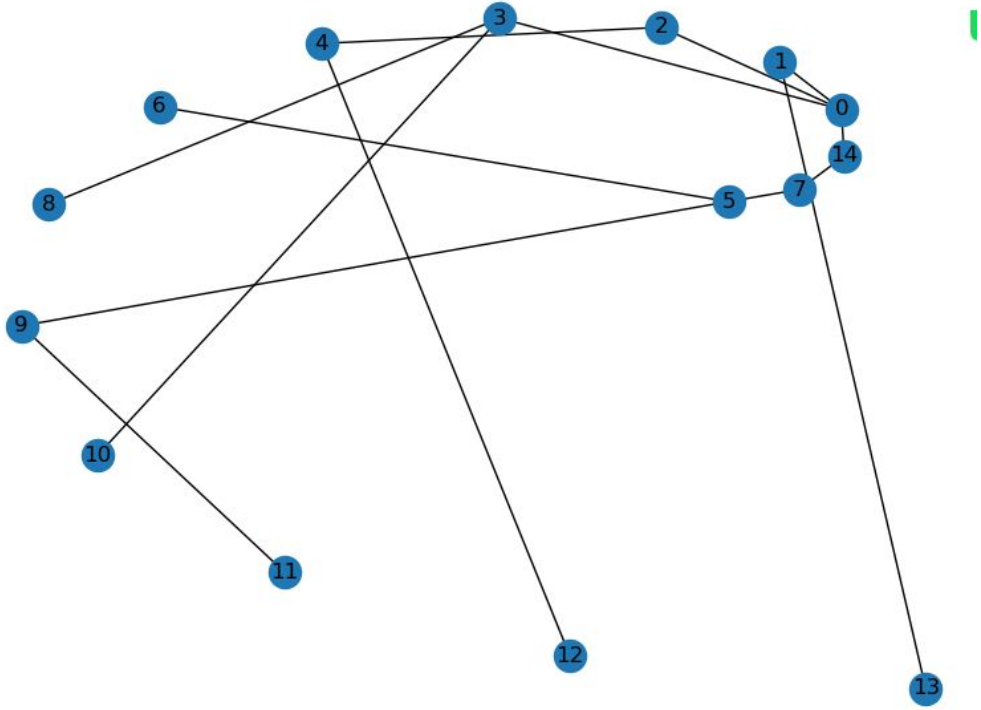
$K = 4, b = 0.001$

Size	Average Degree	Average Clustering	Average Path
30	4.0	10.5	2.3

Barabasi-Albert - Pseudo Code

```
function generateBarabasiAlbertNetwork(vertices, K):
	barabasiAlbertNetwork = createEmptyGraph()
	probabilitiesOfNodes = []  # List of tuples (vertex, probability)

	# Generate initial network with K nodes and at least one edge per node
	selectRandomNode(vertices, barabasiAlbertNetwork)
	counter = K - 1

	while counter > 0:
		selectedNode = selectRandomNode(vertices, barabasiAlbertNetwork)
		addEdgeBetweenNodes(barabasiAlbertNetwork, previouslyAddedNode, selectedNode)
		previouslyAddedNode = selectedNode
		counter = counter - 1
		# Calculate probabilities for nodes in the initial network
		calculateProbabilities(barabasiAlbertNetwork, probabilitiesOfNodes)

	# Connect the remaining vertices to the existing network
	counter = 0
	for node in vertices:
		selectedNode, probability = chooseNodeBasedOnProbabilities(probabilitiesOfNodes)
		counter = counter + 1
		addEdgeBetweenNodes(barabasiAlbertNetwork, selectedNode, node)
		calculateProbabilities(barabasiAlbertNetwork, probabilitiesOfNodes, selectedNode)
		calculateProbabilities(barabasiAlbertNetwork, probabilitiesOfNodes, node)

	return barabasiAlbertNetwork
```

Barabasi-Albert - Graph



Graph Details - Barabasi-Albert

$K = 4$

Size	Average Degree	Average Clustering	Average Path
14	1.8	0.3	3.7

Data Set Graph Details - Original Network

	Original Network			
Network	**Size**	**Average Degree**	**Average Path Length**	**Clustering Coefficient**
Com-Amazon	10000	1.89926	1.98224	0.00808
Twitch Gamers	10000	2.30427	2.76671	0.09808

Data Set Graph Details - Model Graph

	Watts-Strogatz		Barabasi-Albert	
Network	**Average Path Length**	**Clustering Coefficient**	**Average Path Length**	**Clustering Coefficient**
Com-Amazon	3.48305	0.74402	335.58059	0.0
Twitch Gamers	2.48538	0.74710	338.51481	0.0

We used the below command to run the code on CPP's HPC

srun -n 1 -c 4 --mem 100G -p compute -u -J twitchW --output=twitchWatts.log --error=twitchWatts.log python TwitchWatts.py &

srun -n 1 -c 4 --mem 100G -p compute -u -J twitchB --output=twitchBarbasi.log --error=twitchBarbasi.log python TwitchBarbasi.py &

srun -n 1 -c 4 --mem 100G -p compute -u -J amazonW --output=amazonWatts.log --error=amazonWatts.log python AmazonWatts.py &

srun -n 1 -c 4 --mem 100G -p compute -u -J amazonB --output=amazonBarbasi.log --error=amazonBarbasi.log python AmazonBarbasi.py &

Summary

Barabasi-Albert Model:

- Long average path lengths
- Low/ zero clustering coefficients
- Presence of hubs and nodes with low degrees

Watts-Strogatz Model:

- Short average path lengths
- High clustering coefficients
- Achieved by edge rewiring while maintaining local clustering

Original Network:

- Short average path lengths
- High clustering coefficients
- Heterogeneous degree distribution

Model Selection: *Choice depends on desired network characteristics to be replicated.*

Task Performed By Each Member

Name	Task
Mohamed Abdul	Implemented Barabasi Algorithm
Srijit Bhattacharya	Implemented Watts Algorithm
Vikram Ramesh	Computing average degree, average path length, built ring lattice graph.
Gokul Srinath	Built a class to read the data, and constructed the graph.

THANK YOU