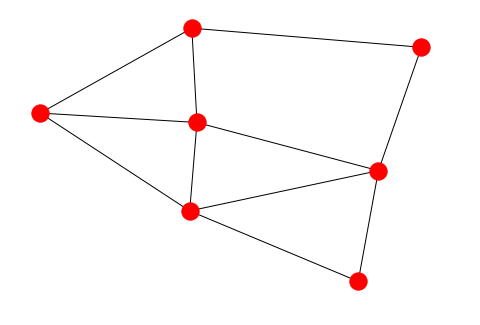
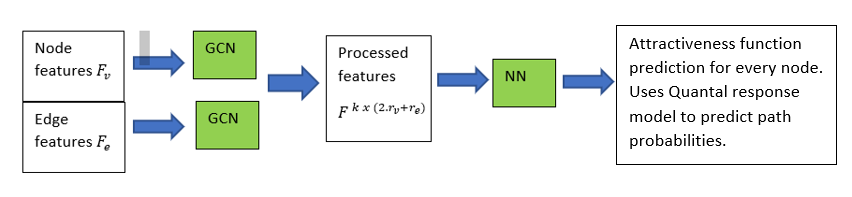
# **Using a CTMC model with GCN for predicting path probabilities in a smuggler network**

# Basic theory and objectives:

  
  
**Assumptions:**Consider the above graph G, with nodes, and an adjacency matrix . Each of the nodes, are assumed to have an associated set of features, captured in the matrix . Similarly, each edge has some features, expressed in the matrix . Our aim is to predict the probabilities of paths taken by the smugglers, for a given defender coverage strategy. So, we assume that the coverage probabilities are also fixed, and known as a vector: , where for some budget



**Computations/Algorithm:**



# What has been done so far:

The algorithm to predict the path probabilities has been implemented using pytorch with the graph shown below.



**Software:**

Two neural networks have been implemented, one to generate synthetic data and other to be used for learning from the data. Each NN consists of a few layers of GCN, that take in node features as input and modify them to return a compressed feature matrix. This is fed to the NN to produce the estimates for the attractiveness function. The coverage probability and the attractiveness is taken into account to calculate the edge transition probability using the quantal response model.   
  
The edge probabilities are then used to compute the path probabilities for all possible paths, by taking product over constituent edges. The GCN and NN parameters are then learnt using pytorch libraries to minimize the MSE loss between the predicted path probabilities and the ground truth path probabilities.

**Results:**

An example run had the following loss values: (after 50 epochs)

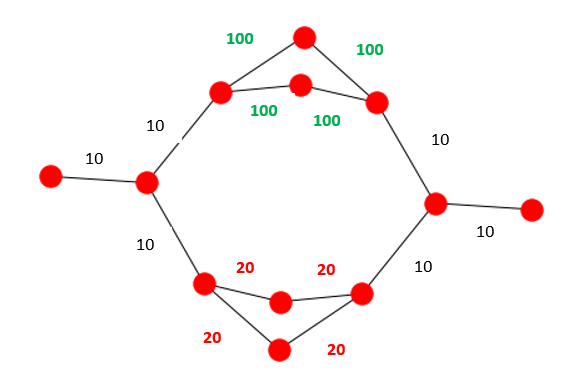
* Loss on testing set before training/after training: 2.3 e-06 / 8.6 e-07
* Training loss before learning/ after training: 2.46 e-06 / 9.2 e-07

**Issues:**1. Currently the loss is calculated over the ground-truth probabilities and the predicted probabilities. However, in real life data, the ground truth probabilities may not be available. We might just have sample paths picked at random using the ground truth probability distribution. Need to upgrade code to learn from sampled path data.

2. Features are randomly generated as of now and do not have any correlation with distance to target/ position of node in graph, etc.

# Issues/Limitations:

## Issue with Markovian approach:



1. Consider the graph shown above. The labeled values indicate the attractiveness of the corresponding edge/next node. So, intuitively the top two paths must have greater probability as compared to bottom two paths, because the edges in middle of the paths are more attractive in the top two paths. However, on tracing the probability calculations, we see that the formula:

So the prob for path1 is actually lower than prob for path 3 using the above calculations!   
  
The root cause of the issue is the term highlighted in red…it doesn’t capture the information that all neighboring nodes are more attractive.   
  
To solve this issue, solution described at the bottom can be adopted:

## Limitations of GCN:

1. GCN has only finite number of layers, so only a finite number of node hops can be computed. So, either the node features must capture non-local properties (which might be counter-intuitive) or we end up with a path probability based on only the neighborhood of the graph.

## Limitations of data:

1. Even though it might seem that it is able to learn the probabilities well just now, **there is a caveat**: The synthetically generated probabilities are obtained by using a GCN too, so the probabilities might not be representative of the real-world dependence of probabilities on the feature set. So, it might be easily possible to fit another GCN that is able to learn this synthetic data and glide over the GCN’s limitation.

# Other ideas/What can be further improved:

1. **ALTERNATE** **MODEL**:

The idea is to model it like an electrical resistance network (This model is equivalent to CTMC model/ water flowing in pipes model/resistor circuit).   
  
Currently, we predict attractiveness for each edge and use it to compute the probability of transition to its neighbors. In this computation however, we ignore the absolute ‘attractiveness’, and essentially use only the attractiveness of a neighbor, relative to other neighbors.

Instead we can use the attractiveness values of edges to compute the overall attractiveness of an entire path and then let the probabilities of paths taken be proportional to path attractiveness. This is like computing the equivalent resistances in an electrical circuit and then dividing the current in that proportion. This seems like an appropriate method of computing probabilities to me.   
In the CTMC model described earlier, the edge attractiveness was just called ‘rate’.

1. Another advantage of the above model is that we do not have to worry about features capturing non-local information. We might as well have local features and use them to predict local attractiveness and then take all such local attractiveness values into account later.
2. The feature set used as of now is randomly generated for every sample point. We could try some non-random features by including features that capture non-local properties (such as distance to target) and check. (Andrew’s idea)  
   This might solve the GCN’s limitation of limited number of node hops, by capturing non-local properties in every node. However, testing this might be hard because the current feature set already seems to be working because the synthetically generated data (as of now) is also generated using a GCN.