Network Software Modelling Assignment 2

1. Describe briefly the real-world phenomenon you wish to model.

War is the phenomenon that forms the basis for our simulation modelling. We thought it would be an interesting subject due to the observable similarities war and network graphs show, and we believed that it would be possible to simulate the evolution of the world map as different factions conflicted with each other during the war.

We found various works done on this topic already. (Dominic Johnson, 2007) used Networks to analyse Civil War in African countries, and features many of the ideas & properties we hypothesized our network to exhibit such as population and GDP, more on this later.

If we think about war, it has the properties of infection – as countries are taken over, they can be considered infected, and this infection spreads quicker as more countries are taken over.

We wanted to model the scenario that a would occur if three factions declared war on another – not dissimilar to the board game ‘Risk’.

If a country wishes to take another, they must fight those connected first, gradually spreading out, as a virus or infection would. A country does not ‘skip’ another one when attacking, rather it attacks countries adjacent to captured ones, that is the neighbours of the node in question.

We liken the chances of infection with the outcome of war. A country will be ‘infected’ depending on the conflict scores achieved. Once conflict is resolved, the attacking country is either successful and takes over the territory (it is infected) or the defending country successfully repels the attack and remains in the state it was in before the attack uninfected).

If, for example, we consider WWII, Germany initially attacked Poland, in order to arrive in a place where they could attack the Soviet Union from. They had to capture the adjacent country (node) in order to progress to the following one, as ultimately, they were interested in capturing the Soviet (not the Polish) land.

The same principles hold true today. While the nature of war has changed since the introduction of manned/unmanned aircraft and more sophisticated marine-based vessels such as aircraft carriers (both of which make it more difficult to model real-world modern warfare), the fundamentals still imply that there must be forces that occupy the ground for it to be considered captured.

If we represent each country with a node, where edges connect it to its adjacent neighbours, we can consider the world a network graph. Alternatively, if a country has no adjacent countries (e.g. Iceland), we may consider the countries closest to it as the nodes it in its neighbourhood, and connected to it. For example, the world may be viewed as the below graphs show:

Code:

2. Describe briefly your choice of graph model and how it corresponds to the real-world phenomenon,

e.g. your choice of directed versus undirected edges, edge weights, allowing or disallowing self-loops,

etc.

As mentioned, we want to emulate the scenario whereby if three factions were to go to war, who would come out on top based on several different factors, which will contribute towards an attack and defence score at each timestep. These in turn will determine the outcome of battle and how the simulation unfolds for a pre-defined number of timesteps.

We introduce properties that were readily available online, relevant to determining the battle outcomes and would contribute to our simulations:

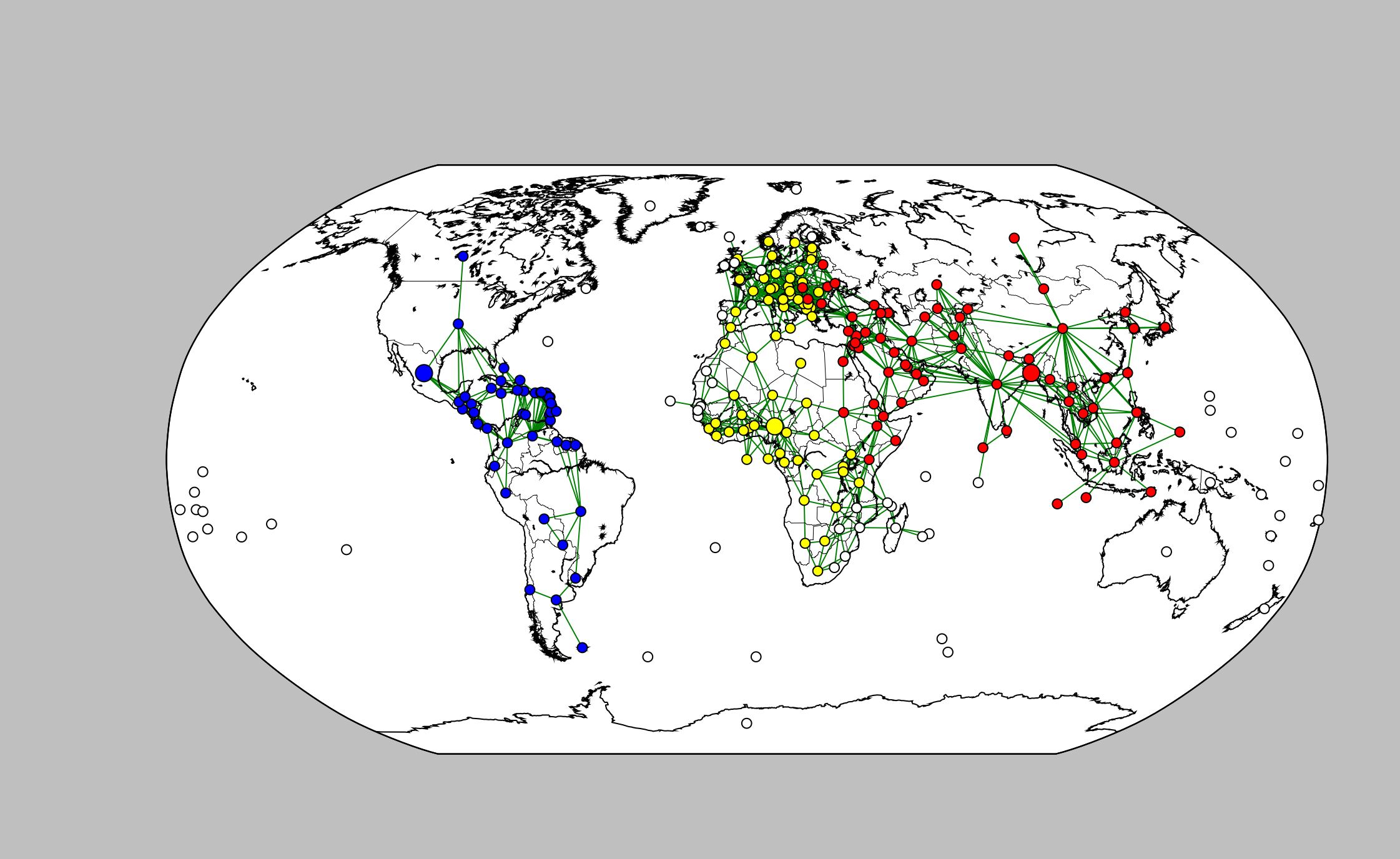
*GDP* This is significant as the higher the GDP, the greater the financial might of the country and the greater the country’s ability to finance war.

*Population* The army will need to consist of enough people to fight the war. The more people, the higher the likelihood of having a talented, well-trained, motivated army at the country’s disposal.

*Distance from Home* It can be assumed that the further away a battle takes place from the home ‘base’ of a country, the tougher it will be for a country to succeed in capturing another country (e.g. USA in Vietnam)

*Random Multiplier* In any conflict, there will be elements of luck that will impact both sides. This factor is included to replicate this luck factor.

Our choice of graph model was the most simplistic we could use in representation of the world geography. It had to be both simplistic, realistic and allow us the required parameters that would be used in the graph evolution. It was limited by the how dense we could make the graph. That is, we had to decide on a tolerance of how far away nodes should be in on graph before creating an edge between them. Choosing too small a tolerance and there would be too many edges, too large and there would be too few. We finally settled on a method that accounts for the population and distance between nodes that provided a good balance and populated our graph nicely with edges. We managed to include the majority of the world’s countries in our graph:



The properties of our graph were as you might expect – it is sparsely diagonally connected. That is, if we consider a binary matrix, where a 1 represents a connection between two countries, and a 0 otherwise, we end up with a very sparsely connected graph. This means that the simulation is not computationally expensive and also realistic in the way that it is less likely countries will be attacking distant ones, at least initially.

It is a directed graph, but it is also dynamic, i.e. the graph’s edges’ direction will change depending on the current Faction. If the Faction owns a territory, the edge connecting it to an adjacent territory is directed – i.e. it can only move that way. Once the next Faction takes its turn, the direction of the edge is reversed (assuming the territory wasn’t taken over of course) and the once defending country can now be invaded.

The **edge weights** are represented by the distance between two countries. This is realistic given the likely rise in importance of a connection, the larger the population. This again means that the graph rarely stays static for very long, evolving as the conflicts get resolved each turn.

**Self-loops** are not a feature of this graph or algorithm as the algorithm points and evolves only towards territories that are not currently under the control of the Faction. It can never choose to attack itself or an allied territory, and so we are unconcerned with self-looping.

The node properties include:

*Geo-Locational coordinates* The latitude and longitude values for the centre of each country to place the node on the map;

*Colour* The colour of each node represents the faction controlling the country;

*Population* Each country has a population property associated with it.

*GDP* Each country has a Gross Domestic Product (GDP) property associated with it.

3. With the aid of a diagram, describe the simulation rules, e.g. how agents change state and what

causes them to send messages.

Now that we have set up our graph with the required nodes and edges, we can begin to discuss how the simulation runs on the graph.

The simulation starts out with three countries representing the three factions – *Red, Blue* and *Yellow* representing each faction, with another colour reserved for neutral countries (*White*).

These countries are randomly selected, but must have at least a certain population before it can be considered a starting point for a Faction. This is where each ‘infection’ originates from.

For each territory (country) under the control of each faction, the country will attack a neighbouring territory if either (a) it is under the control of rival faction or;

(b) it is not owned by the attacking faction or a rival faction (i.e. it has yet to be taken over by anyone).

We also simulated it such that there could only be one attack per Faction before rotating to the next Faction’s attack – more on this later in Section 6.

The order can be simulated or done in a cycle – it shouldn’t be a hugely influential. We chose it to simply take the order of how the nodes were originally added to the graph. Our original data was sorted in terms of GDP and so naturally, the greatest advantage lies with those with the largest GDP’s.

We find the information pertaining to each factor above through online sources. *GDP & Population* are all easily sourced from the internet. The *random multiplier* is calculated per conflict.

A crude model may represent the outcome of a conflict between factions. Such a model may be:

If Population1 > Population2 then

Conflict\_Score = (Population1 / ( Population 1 + Population2) )\* (GDP1/ (GDP1 +GDP2) )

Else:

Conflict\_Score = (Population2 / ( Population 1 + Population2) )\* (GDP2/ (GDP1 +GDP2) )

We determine the outcome of a conflict when we compare the above conflict score with 2 random numbers (between 0 and 1):

If Conflict\_Score > random multiplier \* random multiplier then

Add country 2 to Faction portfolio

Else:

#Do Nothing

The conflicts are resolved one at a time, in each time-step. If a faction is attacking and has a higher conflict score than the territory it is attacking, it is successful and the country being attacked is ‘infected’ and added to the attacking Faction’s portfolio of countries under its control, and is removed from the defending faction’s portfolio . Alternatively, if the attacking faction has a lower conflict score than that of the defending country, no change takes place to either faction portfolio.

The pseudo code may be represented as follow:

Faction\_List ={Red,Yellow,Blue}

For Each Node in Graph:

Faction\_Portfolio ={}#populate the list of countries under the control, of each faction

For each neighbour of Node:

If Faction(Neighbour) == Faction

Else:

#Calcuate both scores

Attack\_score = Conflict\_score

Defense\_score = random multiplier \* random multiplier

If Attack\_score > Defense\_score:

Faction(Neighbour) = Faction

Faction\_Portfolio.append(Neighbour)

Else:

The simulation terminates when either the maximum number of time-steps is exceeded or each node is owned by one Faction i.e. one Faction has conquered the world.

4. Program your simulation in Python. You may use NetworkX, Pregel, Numpy, Scipy, Pandas, Mat-

plotlib, Seaborn, Statsmodels, and the Python standard library. You may also use other libraries for

optimisation, statistics, visualisation, and so on. But if you wish to use any unusual libraries which

relate to graphs or simulation, please contact me first to confirm they are allowed. The main rule will

be that you are required to write your own graph simulation, not to import one written by someone

else.

Code attached

5. State one or more simulation properties which you wish to investigate and which graph properties

you hypothesize they may depend on.

One simulation property that would be of interest investigating would be the average in-degree by network of the graph, and how this affects the eventual outcome of the simulation. We know that the graph is undirected as any country may attack an adjacent one, though each country will have a different in/out-degree depending on where in the world the country is located. This is an important property of the graph, as the higher the out-degree of each country, the higher the amount of attacks it will have, but the more vulnerable it will be to attacks from rival factions. Therefore, we wanted to investigate how the graph would evolve if we set initial conditions such that in one scenario, a faction starts with a country with a small in-degree value (such as in the USA), and in another, it begins with a country with a large in-degree (such as the UK or Germany). Using these (making sure that we normalise all GDP, population, etc. in order to isolate the in-degree effect) simulation conditions will allow us to determine which is optimal. We ran these simulations until we reached a point where we were comfortable with the sample size being large enough to justify the outcome we found. That is, that the \_\_\_ the in-degree, the \_\_\_ likely it is that the country will be successful in world domination. That being said, there was a large element of luck involved and running statistical tests reveals that the average win percentage was not different statistically at a significance of 0.05

We also wanted to investigate what effect it would have if we applied a rule whereby a faction was allowed only one attack before the next faction would have the opportunity to attack. This would theoretically be a fairer simulation and possible more realistic than simple ordering by GDP. We thought that this would result in

Finally we wanted to test out and see what the difference would be if there was a cumulative GDP effect vs.

6. Carry out experiments to measure these properties in different types and sizes of graph (at least one

real-world graph, and at least one scalable graph model such as the Erdos-Renyi random graph model,

at multiple sizes). Present a table of data showing your results.

7. State your conclusions, based on your data.

# References

Dominic Johnson, F. J. (2007). Retrieved from http://privatewww.essex.ac.uk/~ksg/dscw2007/Johnson.pdf