THE REPORT

0. Contents:

1. What I did and What I did not

	Task (in the order of appearance)	Progress	Comments
1.	Dataset Creation	Done	-
2.	Graph Analysis	Not Done	Lack of confidence with my understanding of concepts like centrality(s). It has been a while and as of recent I was more invested in reading papers about Evonet[1], HANConv[2][3], Community Evolution Prediction[4] etc.
3.	Community Detection	Done	-
4.	Link Prediction	Done	-
5.	node2vec: Paper reading task	Done	I am most proud of this. Video is available here: https://drive.google.com/file/d/1j2prLcwfz-OgbyFZimF4dTwGiNdVCzDw/view?usp=sharing

[1]. https://arxiv.org/abs/2003.00842

[2]. https://pytorch-geometric.readthedocs.io/en/latest/generated/torch_geometric.nn.conv.HANConv.html

[3]. https://arxiv.org/abs/1903.07293

[4]. https://www.nature.com/articles/s41598-025-91766-7

2. Introduction

I constructed fully connected graphs for countries, cities and a combined dataset from lists available online, that obeyed the rules of the game of atlas. I only made some preliminary analysis of the data.

I then decided to calculate node embeddings of the graphs using node2vec under different settings. I then used these embeddings in link prediction and community detection tasks.

For link prediction, in the first method, I derived the edge embeddings and trained the prediction model using ConcatMLP. In the second method I used GraphSAGE with HadamardMLP. For ConcatMLP, the only features were the embeddings. For GraphSAGE, the first and the last character formed the feature sets. The former resulted in roughly 80% accuracy on validation sets and the latter resulted in roughly 70% accuracy. The different settings of the embeddings in case of node2vec and the variations in the different datasets caused roughly ±5% variation in the accuracy. In both cases I had used negative samples of the same size as positive samples while training the model. The negative samples here are a permutation of edges that would never occur following the rules of the game of atlas.

For community detection, I first chose to use node2vec embeddings calculated based on BFS strategy, followed by KMeans clustering. The modularity was close to zero, irrespective of whether the dimensions were reduced and convinced that this method under the current settings were not suitable for community detection. Secondly, I chose the Louvain method, which is a greedy randomized algorithm. The modularity on the communities it detected was 0.4, which indicates significant community structure.

3. Data collection and preliminary analysis

To collect the names of all officially recognized countries, as I considered it, my first choice was from Wikipedia[1]. However, for the sake of ease of convenience, I picked listchallenges[2] as the source of information, for this task. The list can be directly copied into a text file for use in code. I picked listchallenges[3] for fetching a list of popular cities as well.

There were 200 entries in the countries list. I manually updated few entries like 'Congo, Republic Of' to 'Republic Of Congo' because I did not want punctuations to be a part of the name and also during a game of atlas, 'Republic Of Congo' is something a player is more likely to mention naturally as opposed to 'Congo, Republic Of'.

The cities list did not require any manual update. However, duplicates existed in the original data of 500 entries and after removing them there were 497 unique entries. The duplicates were ['CORDOBA', 'GRANADA', 'GRAZ'].

Duplicates were found when creating the country+city combined list too. They were: [MONACO, SAN MARINO, SINGAPORE, VATICAN CITY]. Three of these are considered city-states[4] (Monaco, Singapore and Vatican City) and one a microstate[5] (San Marino). So it made sense for them to be listed in both the lists. However, for simplicity, the game of atlas should allow uttering the names of place only once, irrespective of perception as both a country and city. Thus, these entries were considered duplicates and included in the combined dataset only once leaving it with a total of 693 entries.

I decided to use unidecode[6][7] so that I can change the non english characters (like é, ø, ü) from the names of the places to their english equivalent.

3.1. Graph Construction

An edge list together with a node list can be used to represent a graph. The above lists with the names of places of countries, cities and the combined data represents 3 node lists. The graphs corresponding to countries, cities and combined list can constructed along with the edges from the respective node lists.

I began with constructing the countries graph using the countries list (node list). My initial idea was to simulate a game of atlas. I did this by generating a sequence of country names, such that the rules of the game were obeyed, i.e., every consecutive name in the sequence started with the character matching the last character of the previous item in the list. The first name in the sequence always started with the character ‘S’ (because in the the game of atlas, we begin by saying ‘A-T-L-A-S and say a place that starts with S’). The names that followed were picked at random from the list of nodes following the rule and the sequence grew until a duplicate name was produced. 100 such trails produced 100 sequences. The average length of these sequences were around 22. And each consecutive pair in all the 100 sequences were added to a list. After removing the duplicates, this list represented the edge list that would construct the intended countries graph. I had earlier come across a blog on WikiNet[8] where the Wikispeedia[9] data was used. The blog mentions the rules of the Wikispeedia game, “players compete in a race to navigate from a source article on Wikipedia to a set target article by traversing the hyperlinks connecting one page to another”. Although not direct, I found the inspiration to simulate the game of atlas from this blog.

I generated such sequences with the countries list and constructed a countries graph. Some nodes had a degree of zero (in-degree as well as out-degree). For this reason and for reasons mentioned in the link prediction section, I decided to not add this code in the notebook or use the graph for any

further use. And instead, I decided to construct a fully connected graph from the list of country nodes, fully to the extent the rules of the game atlas would allow.

Turns out creating fully connected graph is also computationally less expensive.
For simulation: 100 trials * 22 (average sequence length) * (200 + 497 + 693) = 3058000 units.
For fully connected: $(200)^2 + (497)^2 + (693)^2 = 767258$ units (~ 4 times less).
Table 1. summarizes the number of nodes and edges. In all these cases, 100 trials were conducted for simulations.

Datasets	#nodes	#edges (fully connected)	#edges (simulated)	#simulated edges to #fully connected edges ratio
countries	200	2123	254	~12%
cities	497	10418	1284	~12%
combined	693	21966	1441	~6.5%

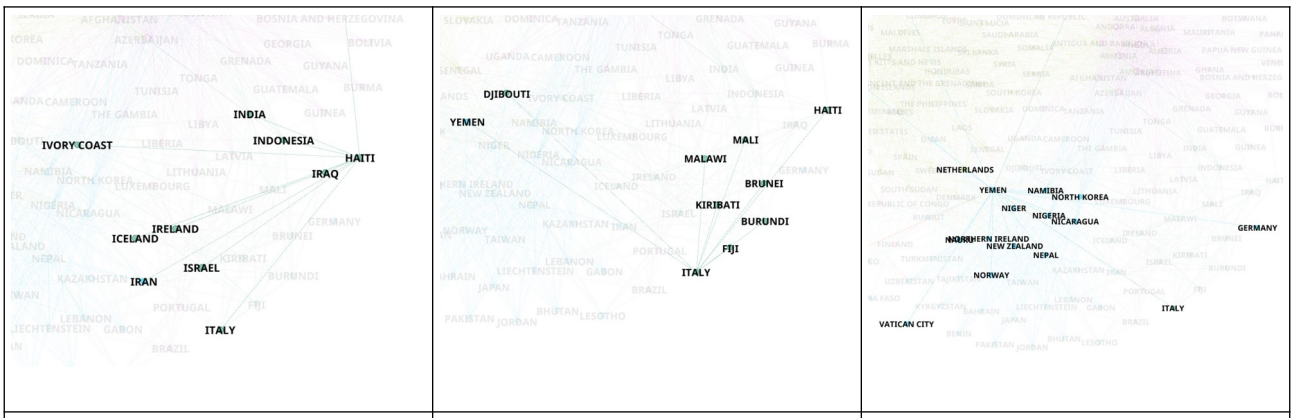
Table 1. Number of nodes and edges

3.2. Node Features

I decided to add some features to the nodes that might be required in the Link Prediction task. Following the task description, I created dataframes, each for countries, cities and combined, with columns for the first and last character of the place name as features. Along with these, I also added the length of the name string, the phonetic key of the name as well as the length of the phonetic key. To get the phonetic key of the names of the places I used the phonetics[10][11] package. Despite being a little outdated in term of machine learning practices, I decided to use phonetics in lieu of word embeddings like BERT[12], due to my lack of familiarity with word embeddings. The phonetics package offered 4 algorithms to calculate the phonetic key for a string. I intended to use Double Metaphone[13] algorithm for this purpose, which is the most recent of all the choices (Metaphone the next most recent choice was 10 years older than Double Metaphone, besides being improved upon). However, I chose the Metaphone algorithm over Double Metaphone. Double metaphone provides a secondary code, in cases of ambiguity. However, in most cases (79% countries, 86% cities and 83% of combined data), there were no ambiguity.

3.3. Preliminary Analysis

Now, there is yet another reason for choosing the phonetic key of the strings as a feature. The simplest strategy, when pondering over the graph analysis task, that occurred to me was choosing the node that has least out-degree among all the options available. For example, if the opponent player says, “HAITI”, the current player would have to say a place starting with “I” (Fig. 1a)). Some available options would be “ICELAND”, “ITALY”, “INDONESIA” and “IVORY COAST”. A good strategy here would be to say, “ITALY”, because it has the least out-degree (out-degree is 1) (Fig. 3b) as opposed to “INDONESIA”, which has an out-degree of 11 or “IVORY COAST” which has an out-degree of 13. And the justification is that, the lesser the out-degree, the less likely the opponent would reach that node. And that is because, it is more likely to have already been mentioned in the game, in some previous rounds. For example, if in some earlier rounds of the game, one of the players have said “GERMANY” which also ends with “Y”(Fig. 3c), the only country in the dataset that starts with “Y”, “YEMEN” could have been said following that, so when the current player now says “ITALY” which has the least out-degree, traps the opponent into forfeiting the game, by doing so. The screenshots in Fig. 1 are taken from the Gephi software[14][15], an open source network analysis tool I used to get some idea about the graphs.



(a) (b) (c)

Fig. 1.

However, based on my personal experience of playing the game (which was more about how well has one memorized the atlas and how many places can one name and thus fun was measured by the longevity of the game), how well can a player recall the name of the place is also a factor. And my hypothesis was that this (how well can a player recall the name of a place), is related to the phonetic key of the name of the place.

Fig. 2 shows a plot of the frequencies of the names of a place ordered by the length of their phonetic key, in all 3 datasets (countries, cities and combined). A large number of the places have a phonetic key length of 3 or 4. So, from here the assumption can be made that a player is more likely to pick the name of a place whose phonetic key length is 3 or 4, based on the distribution. It is little less likely to pink a place whose phonetic key length is 2 or 5. This in fact, supports the aforementioned hypothesis about recall likelihood's correlation with phonetic key length.

However, recency bias is being ignored here. A place, albeit difficult to pronounce, but recently popular in the news would also be reasonably easy for geography enthusiasts to recall. Ignoring this bias, the expected trend would be that the categories of names with phonetic key length of 3 or 4 would quickly get depleted and likelihood of recalling names with phonetic length 2 or 5 raises the longer the game continues. So, at any point of time in the game, based on how long the game has been going on for, we could decide which option among the available ones is likely to have been used earlier and thus avoid it. If we calculate this for 2 degrees, we could also get the likelihood of the node that would make the same difficult for the opponent. Meaning our current player would pick a node, such that all the outgoing nodes from that node have a high likelihood to have been used up earlier in the game provided the players have been playing for a while, and thus trap the opponent into making the mistake of repeating the name of place and thus forfeiting the game.

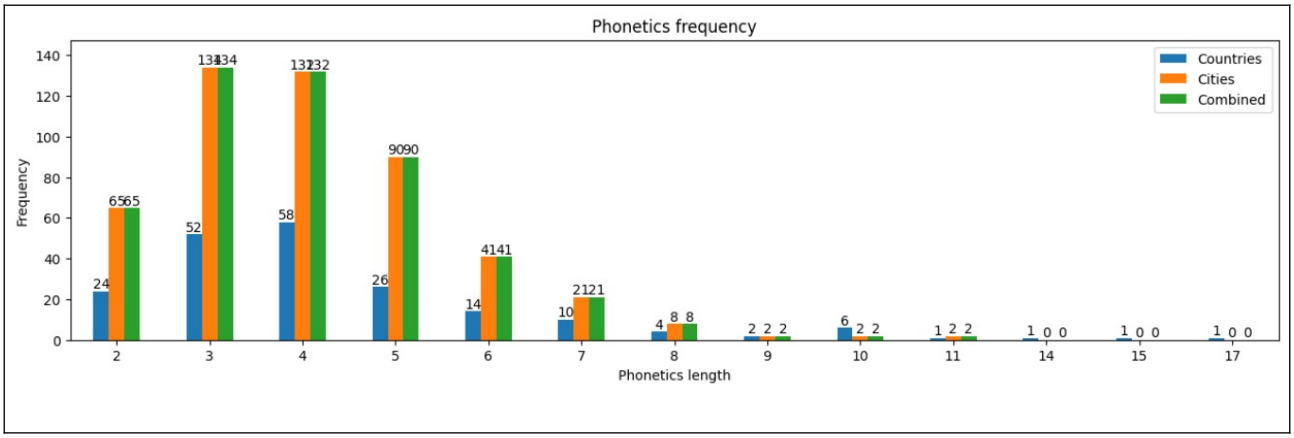


Fig. 2.

However, this was becoming less about graph properties, so I stopped pursuing the idea thereafter.

3.4. Visualization

To, come up with another strategy, I decided to group together all names of places that have the same first character and same last character. For example, “SOUTH AFRICA”, “SOUTH KOREA”, “SLOVAKIA” etc., all have “S” as first character and “A” as last character and therefore be grouped together with the label “SA” representing the first and last characters of all its members. “SWEDEN”, “SPAIN” and “SUDAN” would similarly be grouped together and labeled as “SN”.

The plots in Fig. 3a, b & c are respectively the plots for the graphs with country, city and combined data. I am using the unicode value of the first and last character of the place names as y and x coordinates to position the nodes on the plot. I was intending for this plot (and this idea) to provide some insights on how the betweenness of a group (or overlapping nodes based on their coordinates), when taken together with the strength of the group (number of places/members in a group) could help determine a good winning strategy in the game of atlas. For example, if the opponent says “MALDIVES”, our current player has to say a place starting with “S”, so here the idea is to pick a group say “SN”, “SA”, “SD” or “SE”. The interesting observation is that, all members of a group have identical betweenness centrality, that is, for example, “SWITZERLAND”, “SCOTLAND” and “SWAZILAND”, which are all members of the “SD” group, all have a betweenness centrality of 1.834. Following this example, the proportion of places starting with N,A,D and E are 5.5%, 5.5%, 2.5% and 4.5%. So, among the groups available, any member of “SD” is leading the way to places with “D” as first character. Further exploration can be done here.

[1]. https://en.wikipedia.org/wiki/List_of_sovereign_states

[2]. https://www.listchallenges.com/print-list/251

[3]. https://www.listchallenges.com/print-list/270864

[4]. https://en.wikipedia.org/wiki/City-state

[5]. https://en.wikipedia.org/wiki/European_microstates

[6]. https://pypi.org/project/Unidecode/

[7]. https://github.com/avian2/unidecode

[8]. https://medium.com/stanford-cs224w/wikinet-an-experiment-in-recurrent-graph-neural-networks-3f149676fbf3

[9]. https://snap.stanford.edu/data/wikispeedia.html

[10]. https://pypi.org/project/phonetics/

[11]. https://github.com/ZackDibe/phonetics

[12]. https://en.wikipedia.org/wiki/BERT_(language_model)

[13]. https://en.wikipedia.org/wiki/Metaphone

[14]. https://gephi.org/

[15]. https://en.wikipedia.org/wiki/Gephi

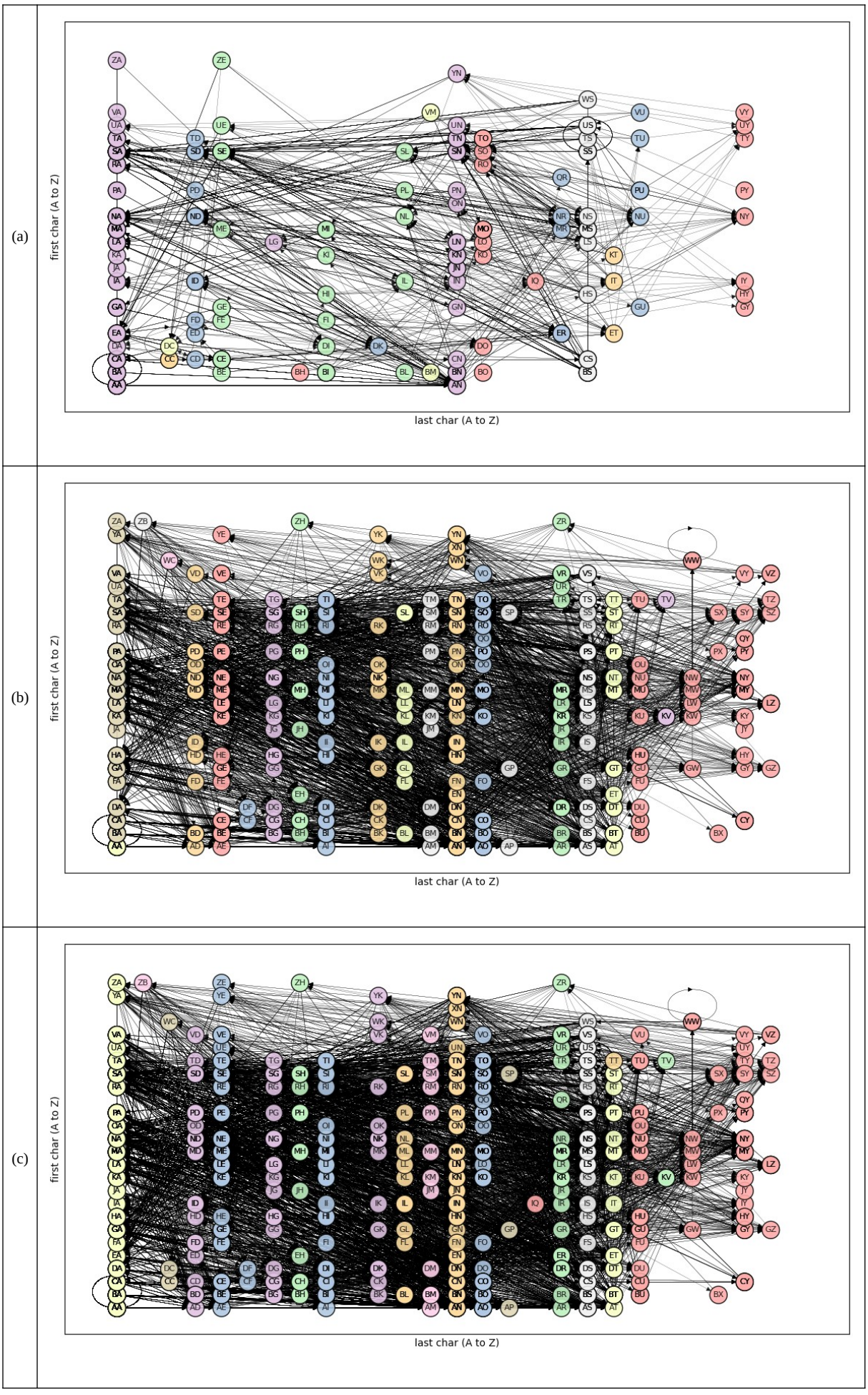

(a)
first char (A to Z)
last char (A to Z)
(b)
first char (A to Z)
last char (A to Z)
(c)
first char (A to Z)
last char (A to Z)

Fig. 3.

4. node2vec embeddings

The paper on node2vec[1] described how the community structure is preserved when the BFS strategy for random walks are used. In addition to this, the paper also mentioned an experiment on finding the missing edges using node2vec embeddings. Based on these points, it made sense to have the node2vec based node embeddings learned and ready for use in community detection as well as link prediction, both of which are a part of this assignment.

I wanted to see, how the variation in the parameters, p and q, leading to differing random walk strategies, would impact the performance of the link prediction model, based on the embeddings learned. And then, I wanted to see how this varied across the different datasets (countries, cities and combined). The combinations I decided to choose for p and q are in the Table 2. These are directly inspired from the node2vec paper. And these values are kept consistent across all three datasets (countries, cities and combined).

Strategy	p	q	Datasets	Model names
DeepWalk[2] (abbr. dw)	1.0	1.0	countries	node2vec_countries_dw
			cities	node2vec_cities_dw
			combined	node2vec_combined_dw
Breadth First (abbr. bfs)	1.0	0.5	countries	node2vec_countries_bfs
			cities	node2vec_cities_bfs
			combined	node2vec_cities_dfs
Depth First (abbr. dfs)	0.5	1.0	countries	node2vec_countries_dfs
			cities	node2vec_combined_bfs
			combined	node2vec_combined_dfs

Table 2. Value choices for parameters p and q

Besides p and q, the node2vec algorithm also requires values parameters for walk length, context size and walks per node to perform random walks. Walk length defines the length of the random walk, context size defines the size of a context which is segmented from the walk (and therefore is less than or equal to walk length) and walks per node defines the number of walks to sample for each node. I will be using the PyTorch Geometric[3] library (also known as PyG) for this implementation and node2vec algorithm is packaged with this library[4].

Based on the aforementioned parameters, it made sense to use average degree, diameter and average path length from a graph to use as number of walks per node, walk length and context size respectively. These values have been calculated from the Gephi and are summarized in Table 3.

Datasets	Avg. Degree	Diameter	Avg. Path Length
countries	10.6	7	3
cities	20.9	5	2.8
combined	31.7	5	2.5

Table 3. Parameters for node2vec algorithm

Convergence when training the model to learn the node2vec embeddings was achieved roughly around 100 epochs and this is consistent across all 9 cases (3 different random walk strategies for each of the 3 datasets: countries, cities and combined as shown in Table 2.). PyG provided an example[5] for the implementation of node2vec on the Planetoid dataset[6]. This was particularly helpful during my implementation of the code.

[1]. https://arxiv.org/abs/1607.00653

[2]. https://arxiv.org/abs/1403.6652

[3] https://pytorch-geometric.readthedocs.io/en/latest/index.html

[4]. https://pytorch-geometric.readthedocs.io/en/latest/generated/torch_geometric.nn.models.Node2Vec.html

[5]. https://github.com/pyg-team/pytorch_geometric/blob/master/examples/node2vec.py

[6]. https://pytorch-geometric.readthedocs.io/en/latest/generated/torch_geometric.datasets.Planetoid.html

5. Link prediction

In order to perform link prediction, we require the link/edge embeddings. So given, the set of all node embeddings, the edge embeddings can be obtained by combining a pair of node embeddings that would represent a possible link/edge. While there are several ways of doing this, I decided to choose these 2 ways: HADAMARD and ConcatMLP.

The node2vec paper mentions that the HADAMARD binary operator preserves the dimension size of the input node embeddings (because it is essentially an element-wise multiplication operation) and among other such binary operations, HADAMARD produced the best results. ConcatMLP, as the name suggests, involves, in this case, the concatenation of the pair of input node embeddings and passing it through a Multi Layer Perceptron. I came across the use of ConcatMLP for link prediction in 2 papers: “ROLAND: Graph Learning Framework for Dynamic Graphs”[1] and “Temporal graph learning for dynamic link prediction with text in online social networks”[2]. The latter utilizes the framework introduced by former and additionally also uses HadamardMLP (their implementation is publicly available[3]). And HadamardMLP is basically a Hadamard operator followed by an MLP.

The link prediction task can be seen as a binary classification task, where given two nodes, we have to determine the likelihood of existence of a link between them. The graph datasets we have provides us information about all the edges that are present (the edge_index) but no true values for a pair of nodes that a model can learn against. So, to overcome that, I had to create a set of negative samples.

My initial approach was to run a simulation to come up with the negative samples. This is similar to how I used simulation to generate a sequence of nodes obeying the rules of the game of atlas to obtain some edges, except here for negative samples the rules would be inverted, i.e., given the name of a place the next name in the sequence would start with any character other than the last character of the preceding name in the sequence. However, as seen in Table 1. for 100 trials, the number of edges produced by this method is less than 15% while being roughly 4 times more expensive to compute. So, instead, for generating negative samples, I decided to create a permutation of edges that would never occur when following the rules of the game of atlas. This is done for all the 3 graphs (countries, cities and combined) in such a way that the number of negative samples match the number edges for a given graph. This balance should help prevent the underfitting of the model.

The positive example (the edges in the graph with true value 1) and negative examples (generated as mentioned above with true value 0) are merged and then a random shuffling of the training, validation and testing masks (with a distribution of 65%, 25% and 10% respectively) was used to split the data. This was repeated for each of the three datasets.

This task requires the use of node2vec and any other GNN to perform link prediction. I chose GraphSAGE[4] for the latter.

5.1. Link prediction using node2vec

The node2vec embeddings learned from the graphs as mentioned in the previous section has been used here. Here ConcatMLP method was used. More precisely, this MLP is composed of one Linear layer followed by a sigmoid activation function. This procedure is summed up in Procedure 1.

Procedure 1. link prediction using node2vec embeddings

Input: edge_index p_e, negative edge samples n_e, node2vec_embeddings H

Output: model for link prediction

```
set y_true to 1 for all edges in p_e and y_true to 0 for all edges in n_e
all_edges ← merge (p_e, n_e)
for each edge in all_edges do:
    edge_embedding ← concat(H(edge[0]), H(edge[1]))
end for
collect train_edges, val_edges and test_edges from all_edges in 0.65, 0.25 and 0.1 distribution
collect y_train, y_val and y_test from y_true corresponding to train_edges, val_edges and test_edges
for 100 epochs do:
    y_train_pred ← model(train_edges)
    model ← backprop(y_train_pred, y_train)
    y_val_pred ← model(val_edges)
    accuracy ← roc_auc(y_val_pred, y_val)
end for
y_test_pred ← model(test_edges)
accuracy ← roc_auc(y_test_pred, y_test)
return model
```

An experiment I decided to run was to see how well each of the 9 sets of node embeddings (Table 2.) would perform link prediction given a pair of node embeddings, once trained against the training set. I decided to run the training for 500 epochs. The validation set was used to measure the performance per epoch. Here, the term performance refers to the accuracy. Accuracy was calculated using the Area Under Curve (AUC) score , particularly the ROC AUC[5] metric provided by the Scikit-Learn library[6][7]. The area under curve measure was mentioned in the node2vec paper when link prediction was being discussed. Also PyG provided a list of tutorials[8] and link prediction[9] was among them. This tutorial used the ROC AUC metric as well. The test set was used to measure the performance post training. Here performance refers to accuracy as well and it was calculated in 2 ways. One is ROC AUC which is same as above. And the other is by measuring the proportion of predicted value (after rounding them close to 0 or 1) that match the true values.

5.2. Link prediction using GraphSAGE

I have followed an example provided by PyG to build the model here[10]. The node embeddings are calculate using 2 layers of SAGEConv[11]. 2 layers in graph convolutions imply that the message passing would take place up to hops for each node. Procedure 2 sums it up. The accuracy measurement is identical to above. This model has then been applied on the 3 datasets (countries, cities and combined).

Some observations were:
1. ROC AUC accuracy measurement was roughly 6% more than the custom accuracy score, in all cases (9 node2vec cases as well as 3 GraphSAGE cases.)
2. The accuracy across the 9 node2vec cases were nearly consistent, in the range of 78% to 86% (AUC).
3. The accuracy across the 3 GraphSAGE cases were also nearly consistent (68%±2%).

Procedure 2. Link prediction using GraphSAGE

Input: nodes_list, edge_index p_e, negative edge samples n_e

Output: model for link prediction

1. set y_true to 1 for all edges in p_e and y_true to 0 for all edges in n_e
2. all_edges ← merge (p_e, n_e)
3. collect train_edges, val_edges and test_edges from all_edges in 0.65, 0.25 and 0.1 distribution
4. collect y_train, y_val and y_test from y_true corresponding to train_edges, val_edges and test_edges
5. for 100 epochs do:
6. calculate node embeddings from nodes_list and edge_list using 2 layers of SAGEConv
7. get edge embeddings using HADAMARD operation for edges in train_edges
8. apply a linear layer with sigmoid activation function
9. back-propagation using y_train
10. validate model using val_edges against y_val and plot ROC AUC accuracy per epoch
11. end for
12. test model using test_edges against y_test and print ROC AUC accuracy
13. return model

[1]. https://arxiv.org/abs/2208.07239

[2]. https://link.springer.com/article/10.1007/s10994-023-06475-x

[3]. https://github.com/manuel-dileo/dynamic-gnn

[4]. https://arxiv.org/abs/1706.02216

[5]. https://en.wikipedia.org/wiki/Receiver_operating_characteristic

[6]. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_auc_score.html

[7]. https://en.wikipedia.org/wiki/Scikit-learn

[8]. https://pytorch-geometric.readthedocs.io/en/latest/get_started/colabs.html

[9]. https://colab.research.google.com/drive/1xpzn1Nvai1ygd_P5Yambc_oe4VBPK_ZT?usp=sharing

[10]. https://github.com/pyg-team/pytorch_geometric/blob/master/examples/graph_sage_unsup.py

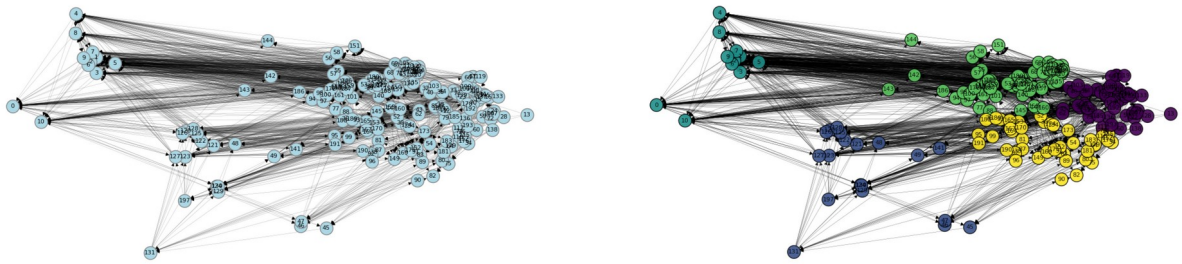
[11]. https://pytorch-geometric.readthedocs.io/en/latest/generated/torch_geometric.nn.conv.SAGEConv.html

6. Community Detection

For community detection two methods were used: node2vec and Louvain method[1][2]. This task was meant to be done on the countries dataset only.

6.1. node2vec for community detection:

I have decided to pick the embedding set that was based on the BFS settings, which is expected to preserve the local properties. These embeddings have a shape of 200(nodes) by 8(dimensions) corresponding to the model named “node2vec_countries_bfs” (see Table 2.). So, I decided to first use PCA[3] to reduce the dimension to 2, so that it can be plotted on a plane surface. Then I used KMeans[4] to cluster the plot into 5 communities. I got inspiration for this from the node2vec paper as well as an implementation I found on github[5]. Fig. 4a and b shows the plot after PCA and after Kmeans respectively.



(a) (b)

Fig. 4

The measure of the strength of division of network into modules (or communities) is called modularity[6]. The Networkx library provides an implementation of this measure[7]. The modularity in the range of 0.3 to 0.7 represents significant community structure[8]. However, with the communities discovered using KMeans with 5 clusters and dimension reduction to 2 using PCA, resulted in a modularity close to zero.

This lead to designing an experiment of using KMeans with clusters ranging from 1 to 26 (the number of alphabets in English was chosen) to fit against both the embeddings (the original 8 dimensional embeddings as well as the 2 dimensional embeddings resulting from PCA). The modularity corresponding to each case was in the range (-0.009 to 0.1).

Based on this it could be pointed that under the settings used in the above experiments and the node2vec parameters (p=1 and 1=0.5), the embeddings could not detect significant communities for the countries graph.

6.2. Louvain method for community detection:

The first option available under community detection in the software Gephi is based on the Louvain method. This is a greedy algorithm and it depends on the ordering of the nodes. The ordering of the nodes is shuffled before running the algorithm so the results may slightly differ each time the algorithm is run on the same data. The Networkx library provides an implementation for this algorithm[9]. Under default settings, this algorithm detects 5 communities and the modularity score is 0.47, which makes this algorithm better over the above implementation of node2vec for

community detection. Table 4. shows the list of communities and its members produced by the Louvain algorithm.

Community	Members
0	'AFGHANISTAN', 'ALBANIA', 'ALGERIA', 'ANDORRA', 'ANGOLA', 'ANTIGUA AND BARBUDA', 'PANAMA', 'PAPUA NEW GUINEA', 'ARMENIA', 'AUSTRALIA', 'AZERBAIJAN', 'AUSTRIA', 'ARGENTINA', 'QATAR', 'ROMANIA', 'RUSSIA', 'RWANDA', 'BOLIVIA', 'BOSNIA AND HERZEGOVINA', 'BOTSWANA', 'BULGARIA', 'BURMA', 'CAMBODIA', 'CANADA', 'CHINA', 'COLOMBIA', 'COSTA RICA', 'CROATIA', 'CUBA', 'TANZANIA', 'THE GAMBIA', 'TONGA', 'TUNISIA', 'UGANDA', 'GEORGIA', 'GHANA', 'VENEZUELA', 'GRENADA', 'GUATEMALA', 'GUINEA', 'ZAMBIA', 'GUYANA', 'JAMAICA', 'KENYA', 'LATVIA', 'LIBERIA', 'LIBYA', 'LITHUANIA', 'MACEDONIA', 'MALAYSIA', 'MALTA', 'MAURITANIA', 'MICRONESIA', 'MOLDOVA', 'MONGOLIA'
1	'VANUATU', 'GREECE', 'PALAU', 'GUINEA-BISSAU', 'ZIMBABWE', 'PERU', 'REPUBLIC OF CONGO', 'BELIZE', 'EQUATORIAL GUINEA', 'CAPE VERDE', 'CENTRAL AFRICAN REPUBLIC', 'CHAD', 'CHILE', 'TUVALU', "COTE D'IVOIRE", 'CZECH REPUBLIC', 'DOMINICAN REPUBLIC', 'EAST TIMOR', 'ECUADOR', 'EGYPT', 'EL SALVADOR', 'ENGLAND', 'MOZAMBIQUE', 'ERITREA', 'ESTONIA', 'ETHIOPIA', 'UKRAINE', 'FRANCE', 'URUGUAY'
2	'POLAND', 'BANGLADESH', 'BELGIUM', 'BRUNEI', 'BURUNDI', 'DEMOCRATIC REPUBLIC OF CONGO', 'DENMARK', 'DJIBOUTI', 'THAILAND', 'DOMINICA', 'TOGO', 'TRINIDAD AND TOBAGO', 'TURKEY', 'FIJI', 'FINLAND', 'VIETNAM', 'HAITI', 'HUNGARY', 'ICELAND', 'INDIA', 'INDONESIA', 'IRAQ', 'IRELAND', 'ISRAEL', 'ITALY', 'IVORY COAST', 'KIRIBATI', 'KOSOVO', 'KUWAIT', 'MADAGASCAR', 'MALAWI', 'MALI', 'MEXICO', 'MONACO', 'MONTENEGRO', 'MOROCCO'
3	'NORTH KOREA', 'NORTHERN IRELAND', 'NORWAY', 'OMAN', 'PAKISTAN', 'PARAGUAY', 'PORTUGAL', 'BAHRAIN', 'BENIN', 'BHUTAN', 'BRAZIL', 'BURKINA FASO', 'CAMEROON', 'TAIWAN', 'TAJIKISTAN', 'TURKMENISTAN', 'GABON', 'UZBEKISTAN', 'GERMANY', 'VATICAN CITY', 'YEMEN', 'IRAN', 'JAPAN', 'JORDAN', 'KAZAKHSTAN', 'KYRGYZSTAN', 'LEBANON', 'LESOTHO', 'LIECHTENSTEIN', 'LUXEMBOURG', 'NAMIBIA', 'NAURU', 'NEPAL', 'NEW ZEALAND', 'NICARAGUA', 'NIGER', 'NIGERIA'
4	'BAHAMAS', 'BARBADOS', 'BELARUS', 'SAINT KITTS AND NEVIS', 'SAINT LUCIA', 'SAINT VINCENT AND THE GRENADINES', 'SAMOA', 'SAN MARINO', 'SAO TOME AND PRINCIPE', 'SAUDI ARABIA', 'SCOTLAND', 'SENEGAL', 'SERBIA', 'SEYCHELLES', 'SIERRA LEONE', 'SINGAPORE', 'SLOVAKIA', 'SLOVENIA', 'SOLOMON ISLANDS', 'SOMALIA', 'SOUTH AFRICA', 'SOUTH KOREA', 'SOUTH SUDAN', 'SPAIN', 'SRI LANKA', 'SUDAN', 'SURINAME', 'SWAZILAND', 'SWEDEN', 'SWITZERLAND', 'SYRIA', 'CYPRUS', 'COMOROS', 'THE PHILIPPINES', 'UNITED ARAB EMIRATES', 'UNITED STATES', 'WALES', 'HONDURAS', 'LAOS', 'MALDIVES', 'MARSHALL ISLANDS', 'MAURITIUS', 'NETHERLANDS'

Table 4.

[1]. https://en.wikipedia.org/wiki/Louvain_method

[2]. Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, Etienne Lefebvre, Fast unfolding of communities in large networks, in Journal of Statistical Mechanics: Theory and Experiment 2008 (10), P1000

[3]. https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html

[4]. https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

[5]. https://github.com/Anika-Roy/Exploring_Graph_Embeddings/blob/main/node2vec_Les_Miserables.ipynb

[6]. https://en.wikipedia.org/wiki/Modularity_(networks)

[7]. https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.community.quality.modularity.html

[8]. https://www.youtube.com/watch?v=mJQrtXZT5pw

[9]. https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.community.louvain.louvain_communities.html

End of report