Homework 3
Significance of Metrics

Juan Pablo Royo Sales

Universitat Politècnica de Catalunya

October 31, 2020

Contents

1 Introduction

In this work I have been analyzing the significance of Closeness Centratlity Measure (Closeness Centratlity) metric over a Provided Language Set (Language Set) provided for such case. Basically the idea of the work is the following: Giving this set of Languages with some set of connected terms, determine the Closeness Centratlity accurately measured by:

$$C = \frac{1}{N} \sum_{i}^{N} C_i \tag{1}$$

where,

$$C_i = \frac{1}{N-1} \sum_{j \neq i}^{N} \frac{1}{d_{ij}} \tag{2}$$

After that I am going to measure the **Significance** of that metric against different Null Hypothesis, in particular against Null Model based on Switching (Switching Model) and Null Model based on Binomial Erdos Renyi (Binomial Model).

Using a Montecarlo Approximation Algorithm (Montecarlo) I am going to approximate the *p-value* of both **Null** Hypothesis against the C for each language.

The present work is divided in the following sections:

- **Results** section: Where I show the results obtained for each language according to the previous statements.
- **Discussion** section: In this section I will provide some discussion about the results and my personal impression.

- **Methods** section: Where i am going to describe each of the method and technique use to do the work.
- **Conclusions** section: Finally I am going to give my opinion about the results obtained and what I have learnt.

2 Results

Lets divide the results for each language, but first lets see the Summary of the data that I am analyzing.

2.1 Languages Main Characteristics

Language	N	E	$< k >$	δ
Arabic	21532	68743	6.3851941	0.0002966
Basque	12207	25541	4.1846482	0.0003428
Catalan	36865	197075	10.6917130	0.0002900
Chinese	40298	180925	8.9793538	0.0002228
Czech	69303	257254	7.4240365	0.0001071
English	29634	193078	13.0308430	0.0004397
Greek	13283	43961	6.6191372	0.0004984
Hungarian	36126	106681	5.9060510	0.0001635
Italian	14726	55954	7.5993481	0.0005161
Turkish	20409	45625	4.4710667	0.0002191

Table 1: Language List - Main Characteristics

2.2 Closeness Centrality Measures

Language	Metric	p-value (switching)	p-value (binomial)
Arabic	0.3264629	1.00	0.00
Basque	0.2697357	0.00	0.00
Catalan	0.3410137	1.00	0.00
Chinese	0.3264533	0.00	0.00
Czech	0.3059504	0.392	0.00
English	0.3435140	0.00	0.00
Greek	0.3147265	1.00	0.00
Hungarian	0.2883451	1.00	0.00
Italian	0.3278253	1.00	0.00
Turkish	0.3603387	0.00	0.00

Table 2: Language List - Closeness Centrality

3 Discussion

3.1 Switching

Accordingly to the results obtained as it can be appreciate in 2 the majority of the **p-value**, it means $\frac{6}{10}$ for the Switching Model are greater than 0.05. This means that I can **accept the Null Hypothesis (Null Hypothesis)** for the Switching Model in the case of those languages. As we can see for almost all iterations Closeness Centratlity value is greater than the real model C. For detailed iterations and values of each Closeness Centratlity of each iteration and for each language, please see results on `results/results_montecarlo_XXX_1.txt` where XXX is the raw results for language XXX of the Montecarlo, where it is describe the result for each iteration.

Lets analyze what happen for those cases that I **cannot accept** Null Hypothesis because the *p-value* is 0.00. For doing that I will purpose to analyze each of those rejected language against some language whose *p-value* **has been accepted** and it has similar characteristics. So lets define Similar.

Given the set of languages $L = \{Arabic, \ldots, Turkish\}$ A language S_i is similar to other language if the difference in number of vertices are minimum with respect to L:

$$S_i = \underset{j \in L, j \neq i}{\arg\min} \{|N_j - N_i| \wedge p_{value}(j) > 0.05\} \tag{3}$$

- **Basque**: $S_{Basque} = Greek$. In this case we can appreciate that $< k >$ and δ in **Greek** is higher which indicate that the probability of being connected with central nodes are higher. This can explained why Switching Model preserves the structure and centrality in this case, which is more difficult to achieve with **Basque**.
- **Chinese**: $S_{Chinese} = Catalan$. In this case we observe the same as in the previous case of **Basque**. This language **Chinese** has higher $< k >$ and δ rather than **Catalan** which is the most Similar with acceptance of Null Hypothesis.
- **English**: $S_{English} = Hungarian$. In this case since the $< k >$ is extremely high in comparison to the density this indicates that although **English** has a higher degree on average, that does not mean are dense, which in random permutations those start to be lost for the number of terms and edges.
- **Turkish**: $S_{Chinese} = Catalan$. In the case of **Chinese** we can appreciate that the δ is lower than **Catalan** which could indicate the same as **English**.

3.2 Binomial

Regarding the Binomial Model none of the random generated graphs has been close to the real Closeness Centratlity of the given models, and it can be seen by the results output file `docs/results_montecarlo_XXX_2.txt` where XXX is the raw results for language XXX of the Montecarlo, that all of the them behaves in a stable number according to the number of **Edges and Vertices** that were provided for generating the Random Graph.

This could be explain by the fact that Binomial Model are fully random graph and Closeness Centratlity is an intrinsic property of **non random graph**, because it indicates how concentrated are the elements around some of them. In the previous Switching Model this behavior was different and it gives more accurate Closeness Centratlity values, because it is a permutation over the same graph, and in that permutation I tried to preserve the structure of the original graph, although the selection of the **vertices** that are selected for switching itself is random.

Therefore, I need to **reject** the Null Hypothesis in the case of Binomial Model.

4 Methods

In this section i am going to describe the method used for arriving to this conclusions, starting from analyzing the source code, following up to the parameters selected for running the analysis.

4.1 Source Code Implementation

When we talk about the methodology that it is used for conducting this kind of experiments it is a very important to remark all those decisions regarding the implementation in the source code. As it can be seen in the folder of this distribution, the language use was **C++**.

4.1.1 Graph Representation

Basically the Graph of the language is represented by an Adjacency matrix of the **Vertex** and **Edges** that are connected to those edges.

```
class Graph
{
    string language;
    int V;
    int E;
    vector<int> *adj;
```

Listing 1: Extracted from source code graph.cc

All the computations are done over the array of edges `vector<int> *adj` which is the adjacency list.

4.1.2 Order of Edges

After several and different experimentation I have finally decide to order the edges by **Degree sequence in descendent order**. This order is done just before calculating the Closeness Centratlity to speed and optimize the process. In fact **Adjacency list of nodes are not sorted at all**, the sorting process is done *on-the-fly* when we need to iterate over the vertices for calculate Closeness Centratlity.

```
double Graph::ClosenessCentrality(){
    double closeness = 0.0;
    int **distances = new int*[V];
    vector<bool> distCalculated;
    distCalculated.assign(V, false);
    for (auto i: sort_indexes(adj, V)) {
        closeness += Closeness(i, distances, distCalculated);
    }
    delete *distances;
    return closeness/(1.0*V);
```

Listing 2: Extracted from source code graph.cc

4.1.3 Closeness Centrality

The calculation of Closeness Centratlity is as it was proposed in the work, a regular **BFS** Bread First Search algorithm that can be seen here:

```
void ClosenessBFS(int s, int **distances, vector<bool>
↪  distCalculated)
```

Listing 3: Extracted from source code graph.cc

In order to **speed the process** I have also implemented the proposal of the work to calculate for those nodes whose $k < 2$ taking the distance of the parent $+1$.

```
double Closeness(int s, int **distances, vector<bool>
↪ distCalculated){
    if(adj[s].size() == 0) return 0.0;
    if(adj[s].size()<2){
        ClosenessLowDegree(s, distances, distCalculated);
    }else{
        ClosenessBFS(s, distances, distCalculated);
    }
    double closeness = 0.0;
    for(int i = 0; i<V; i++){
        if(distances[s][i] > 0) closeness +=
        ↪ 1/(1.0*distances[s][i]);
    }
    return closeness/(1.0*(V-1));
}

void ClosenessLowDegree(int s, int **distances,
↪ vector<bool> distCalculated){
    if(!distCalculated.at(s)){
        int node = adj[s].front();
        distances[s] = new int[V];
        for(int i = 0; i<V; i++) distances[s][i] = 0;
        if(adj[node].size()<2 && adj[node].front() == s){
            distances[node] = new int[V];
            distances[s][node] = 1;
            distances[node][s] = 1;
            distCalculated.assign(node,
            true);
        }else{
            CopyDistancesPlus1(node, s, distances);
        }
        distCalculated.assign(s, true);
    }
}
```

Listing 4: Extracted from source code graph.cc

Also I have implemented the **Optimization** regarding to not calculate the whole Closeness Centratlity for the whole graph, and only a portion of them taking and $M = \frac{N}{1000}$.

```
double Graph::ClosenessCentralityReduced(int M){
    double closeness = 0.0;
    int **distances = new int*[V];
    vector<bool> distCalculated;
    distCalculated.assign(V, false);
    int a = 0;
    for (auto i: sort_indexes(adj, V)) {
        if(a >= M) break;
        closeness += Closeness(i, distances, distCalculated);
        a++;
    }
    return closeness/(1.0*V);
}
```

Listing 5: Extracted from source code graph.cc

4.1.4 Montecarlo

The main algorithm can be found here:

```
double CalcPValue(){
    int countSuccess = 0;
    int M = graph.GetV()/1000;
    double closenessOriginal =
    ↪  graph.ClosenessCentralityReduced(M);
    for(int i = 0; i < T; i++){
        if(calcCC(closenessOriginal, M)) countSuccess++;
    }
    return countSuccess/(1.0*T);
}

private:

bool calcCC(const double closenessOriginal, const int M){
    Graph* newG;
    if(model == 1){
        newG = DoSwitching();
    }else{
        newG = DoBinomial(graph.GetV(), graph.GetE());
    }
    double closeness = newG->ClosenessCentralityReduced(M);
    cout << "Closeness c_mix_hnull " << fixed <<
    ↪  setprecision(7) << closeness << " - original "  <<
    ↪  fixed << setprecision(7) << closenessOriginal <<
    ↪  endl;
    return closeness >= closenessOriginal;
}
```

Listing 6: Extracted from source code aprox.cc

In the case of Montecarlo I have implemented the generation of the Binomial Model which can be found here:

```
Graph* DoBinomial(int V, int E)
{
    Graph* g = new Graph(graph.GetLanguage(), V);
    int i, j, edge[E][2];

    i = 0;
    // Build a connection between two random vertex.
    while(i < E)
    {
        edge[i][0] = (((double) rand() / (RAND_MAX)) * V)
        ↪  +1;
        edge[i][1] = (((double) rand() / (RAND_MAX)) * V)
        ↪  +1;

        if(edge[i][0] == edge[i][1])
            continue;
        else
        {
            for(j = 0; j < i; j++)
            {
                if((edge[i][0] == edge[j][0] && edge[i][1]
                ↪  == edge[j][1]) || (edge[i][0] ==
                ↪  edge[j][1] && edge[i][1] ==
                ↪  edge[j][0]))
                    i--;
            }
        }
        i++;
    }
```

Listing 7: Extracted from source code aprox.cc

There are more lines that it is not appearing here 7 because to fit in the page.

And in the case of Switching Model it is a slightly more complicated because i needed to take care of no breaking the structure of the graph when i toss the coin to switch 2 terminals of 2 different edges.

Basically the main implementation of the Switching Model can be seen here:

```
Graph* DoSwitching(){
    Graph* newG = new Graph(graph);
    for(int i = 0;i<Q*newG->GetE();i++){
        int randomVertex1 = ((double) rand() / (RAND_MAX))
        ↪  * newG->GetV();
        int randomVertex2 = ((double) rand() / (RAND_MAX))
        ↪  * newG->GetV();
        if(randomVertex1 != randomVertex2){
            newG->TrySwitch(randomVertex1, randomVertex2);
        }
    }
    return newG;
}
```

Listing 8: Extracted from source code aprox.cc

But the `TrySwitch` method which is doing the whole work can be found here:

```
void Graph::TrySwitch(int v1, int v2){
```

Listing 9: Extracted from source code graph.cc

There are more lines that it is not appearing here 9 because to fit in the page.

4.2 Optimization and Improvements

I have done both optimizations proposed by the work:

- Not calculating exhaustively Closeness Centratlity for nodes whose degree is less than 2 $k < 2$. This has been shown here 4
- In the case of Montecarlo Calculate a portion M for both the real model and the Null Hypothesis. This has been explained here 5

4.3 Experiment Parameters

Implementing all the optimizations described before 4.2, I was able to produce experiments with higher value of T for all languages in the case of Switching Model.

- $T = 1000$

- $Q = 15$

For the case of Binomial Model since i detected early that because of the fully randomization i will never going to achieve a close result, i decided to setup in the minimum value which is 100.

- $T = 100$

All the outputs of the experiment with each value generated can be found as i pointed out at the beginning of this document `results/results_montecarlo_XXX_1.txt`, suffix 1 for the case of Switching Model and 2 for Binomial Model.

5 Conclusions

One of the first conclusions i can arrive after doing these experiments is that Switching Model is a more accurate tool in order to measure the significance of Closeness Centratlity. On the other hand Binomial Model is something that in the case of Closeness Centratlity doesn't apply because of the fully randomization of the generated model, which cannot represent properly a strong relationship regarding a set of important nodes.

Moreover, it can be appreciated that perhaps regardless the value of T Binomial Model is not suitable as measuring tool for this metric.

As a future work, it would be important to explore other mechanisms to tunning better the Null Hypothesis model generators in order to have the same level of randomization but preserving better Closeness Centratlity characteristics of the original model.

In conclusion although it is quite hard to iterate over a large number of T in order to measure properly the Closeness Centratlity based on the Null Hypothesis, it has been seen that Switching Model gives us an accurate approximation.

A Source Code and Assets

The source code and different assets that are contained in these folders are classified in the following way.

- **bin** This folder contains the compiled binary files for the code that implements the requirements suggested by the lab work.

- **data** This folder contains the input Language Set provided.
- **docs** This folder contains this report in Latex and PDF compiled format.
- **out** This folder contains the standard output results after executing each program over the Language Set
- **results** This folder contains the standard output of the raw results obtained in the experiments conducted for each Language Set
- **src** This folder contains the `cpp` files with the source code implementation.
- `./Makefile` Makefile to compile programs
- `./README.md` Markdown file with the instructions to run the program.
- `./run_closeness.sh` Script file for running the algorithm over the Language Set in order to calculate *Closeness Centrality* and main features of each language.
- `./run_montecarlo.sh` Script file for running the approximation algorithms for each language with null models.

B Running and Compiling

B.1 Compiling

```
> make clean
> make all
```

All the binaries are going to be place in `bin/` directory.

B.2 Help

```
> bin/closeness -h
> bin/montecarlo -h
```

B.3 Running for a Single Language

B.3.1 Closeness Centrality for real Model

```
> bin/closeness "data/Basque_syntactic_dependency_network.txt"
```

B.3.2 Montecarlo Approximation for Switching Model

```
> bin/montecarlo "data/Basque_syntactic_dependency_network.txt" T Q 1
```

Where T is the number of iterations of Montecarlo Method and Q the number of Switching that is going to be multiplied by the number of Edges to get the total number of switching.

B.3.3 Montecarlo Approximation for Binomial Model

```
> bin/montecarlo "data/Basque_syntactic_dependency_network.txt" T Q 2
```

B.4 Running for ALL Language

```
> sh run_closeness.sh
```

```
> sh run_montecarlo.sh
```

WARNING: This script might take more than 6 HOURS TO FINISH.