# Making Graphs Great Again

An insight over phrase prediction using graph models

Igor Neres Trindade
Federal University of ABC
Santo André, Brazil
igor.neres@aluno.ufabc.edu.br

Victor Henrique Silva de Oliveira
Federal University of ABC
Santo André, Brazil
victor.henrique@aluno.ufabc.edu.br

*Abstract* — **Studying the communication, more specifically the language and its structure, is very important for many different applications in several devices. Many existing technologies are already able to have some understanding of a given text message.**

**As communication is very complex, there are distinct strategies and objectives when embracing the challenge of understanding a message. In this project, we have the specific goal of getting an insight over the understanding of a text and phrase prediction.**

**For that, we used graph modeling, running algorithms relating the words that make up the frequent vocabulary of an individual, in order to study the relation between the words and the individual's pattern of behavior and personality.**

**A program in *R* language that predicts words and creates phrases according to user inputs came up as a result of many analysis of the modeled graph. Manifold insights over word prediction using graph emerged from this research**

***Keywords — graph theory; Gephi; R; word prediction.***

## I. Introduction

We live in a moment of digital disruption, constant innovation, and emerging digital technologies, an era of instantaneity, hurry and all that for development. *Time is money*, people say today. In this scenario, many applications (most of them for web and mobile) were created to assist people in faster actions, communication included, as assistance on feedbacks (automatic suggestion of response according to a message), dynamic read of some text, etc.

However, the communication is a very complex process. And, indeed, scientists and developers must do a huge effort to reproduce this process, that is natural for us, social beings, on computers. With the development of cognitive learning and artificial intelligence, the computer acquires the ability to learn from data it processes.

Studying how words can be related based on a standard of behavior is intrinsically important. It is relevant, for example, to the enhancement of services that want to know the behavior of their consumers. Besides that, knowing the frequency that words appear after others, we can predict human intentions and suggest what word probably could come after the other and create phrases. Approaches from different studies may enhance artificial intelligence tools as well.

The main challenge this project embraced was to understand better how word prediction and phrase building works. In few words, we extracted data from a real textbook and modeled it as a graph, using the Graph Theory. Our main goal was to create an algorithm that synthesizes the main idea following each word. We also analyzed how efficient the algorithms were, according to the behavior of the graph.

Graphs have a large range of applications. And it fits very well for this purpose. Future enhancement will be provided for this work and all the content generated is shared as an open-source content, available in the GitHub page *https://github.com/igornerest/Making-graphs-great-again*

## II. Some advances in this area

Among the manifold advances in this area, we would like to highlight three specific programs developed. They all have different approaches but work in the same sense of having a better understanding of the communication.

The first one, that is present in the life of most people, is a set of applications by Google. A helpful tool in actual smartphones developed by this company, for example, can predict words while someone types the keyboard. The keyboard learns from the users and, from this knowledge, that is constantly being improved, it can anticipate words (*Fig. 1*). More than that, it can correct mistyping, translate sentences to another language or even write a sentence from a voice typing.

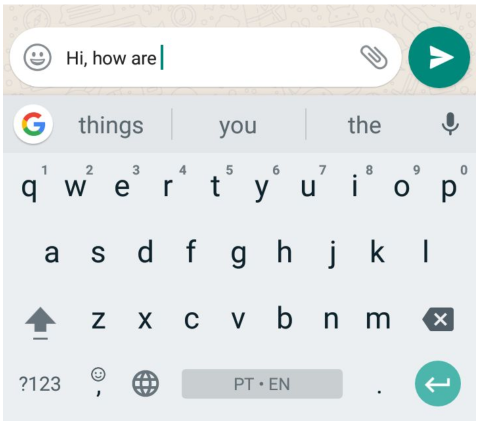
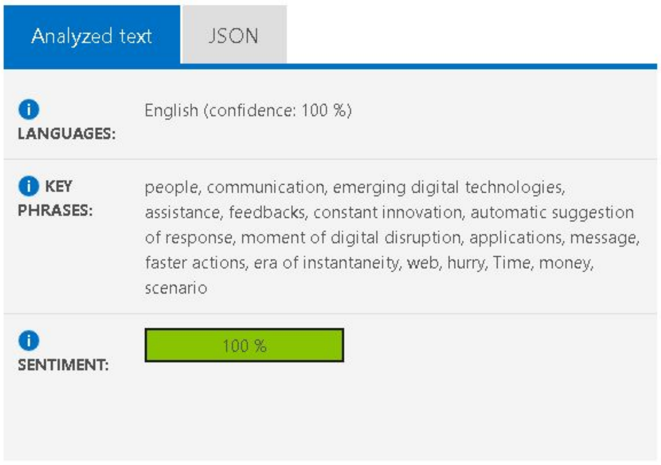


*Fig. 1. GBoard - keyboard anticipating words while typing a message in Whatsapp Messenger app*

Other applications are faced for more specific objectives. *Grammarly*, for example, is a grammar checker and can correct message in real time in almost every web application, through the browser extension. It also works with constant learning, enhancing its knowledge from the use by different people.

Finally, we must consider *Microsoft Cognitive Services*. In this case, it's important to mention the text analytics application programming interface (API). It uses an algorithm to understand the sentiment behind a text and returns some key phrases. It has a ton of applications, and one of them is for the e-commerce: very often, an online shopping cannot read all the critics about its products.In *Fig. 2*, we analyzed the first paragraph of the introduction to see how it works.



LANGUAGES: English (confidence: 100 %)

KEY PHRASES: people, communication, emerging digital technologies, assistance, feedbacks, constant innovation, automatic suggestion of response, moment of digital disruption, applications, message, faster actions, era of instantaneity, web, hurry, Time, money, scenario

SENTIMENT: 100 %

*Fig. 2. Text analytics API by Microsoft. We tested the first paragraph of the introduction.*

In the midst of so many possibilities, in this project, we focused in the word prediction.

## III. Methodology

A text can be represented by graphs, nodes represent words, edges represent the connection between words. As we are concerned in phrase building, this graph, in particular, must have directed edges. We managed to extract data from an epub file of the book *Crippled America: How to Make America Great Again*, written by Donald J. Trump, after converting it to a text file and being able to read it in a Java program. Our program wrote, line by line, each directed connection between words in a new text file, using a spreadsheet format, so we should be able to load it into Gephi and understand better the relation between words.

Gephi was not only used to plot graphs and calculate important metrics. It was also an important step to create other spreadsheets (with a filtered data), that we could not create through the Java program. With these new tables in a proper format, using R programming tools, we were able to develop a program that, analyzing the whole graph, and reading some user inputs, could create phrases according to a desired number of words.

### A. *Extracting Data and Modeling*

The data was extracted from the book *Crippled America: How to Make America Great Again*, written by Donald Trump. The book presents his blueprints for how to make the United States prosperous, as also his thoughts and ideas on some controversial topics.. As he says in the text, *we just need someone with the courage to say what needs to be said*. We expected that the book synthesized the main characteristic of Trump's speeches and opinions

The graph was modeled as a directed connection with a single type of vertex. Words were modeled as vertices and the connection between them as edges. It's important to notice that the ordering is important. Let's use a phrase from the book to exemplify

*It's time to bring America back to its rightful owner - the American people*

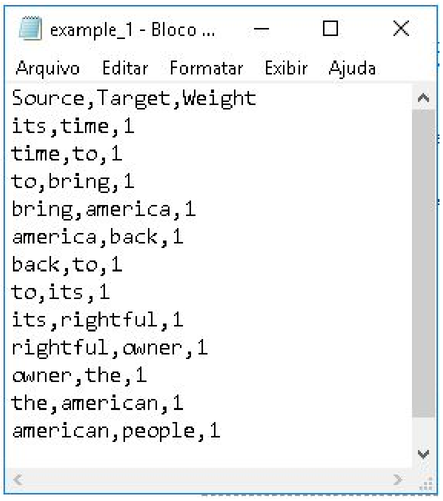
Let the graph G = (V, E) be an ordered pair, then we should have:

- **Set of vertices V** = { its, time, to, bring, America, back, to, its, rightful, owners, the, American, people}
- **Set of edges E** = { {its, time}, {time, to}, {to, bring}, {bring, America}, {America, back}, {back, to}, {to, its}, {its, rightful}, {rightful, owners}, {owners, the}, {the, American}, {American, people} }

An important step of our abstraction was considering the essence of the phrases, i.e, we ignored punctuations and simplified some structures. *It's,* for example, was simplified to *its.*

In order to create a graph, we should somehow store all the data related to the graph structure in a document able to be loaded to Gephi, R or any other processing program. For that, we chose the node table and edge table file formats. They are used in the Data Laboratory, in Gephi, to import data from Excel. But text files presenting the same structure work as well.

The structure is simply composed of several lines. Each line describes a node - with an *id* , in the case of a node table - or an edge - with columns representing the *source* node, the *target* node and the *weight* of the edge, in the case of an edge table -. Using the same example as before, we would have a text document as shown below (in *Fig. 3)*



```
Source,Target,Weight
its,time,1
time,to,1
to,bring,1
bring,america,1
america,back,1
back,to,1
to,its,1
its,rightful,1
rightful,owner,1
owner,the,1
the,american,1
american,people,1
```

*Fig. 3. Example of an edge table in .txt file format*

To create an edge table like the one above, we used a program, that read the whole textbook in in a text file format and created, from the book data, the expected document

## *B. Processing Data with Java program*

A Java program was created to synthesize the whole textbook into an edge table. For each node, i.e, for each *id* associated to a node to represent words, only letters were captured. Any other symbols were ignored when creating an edge table in a text file.

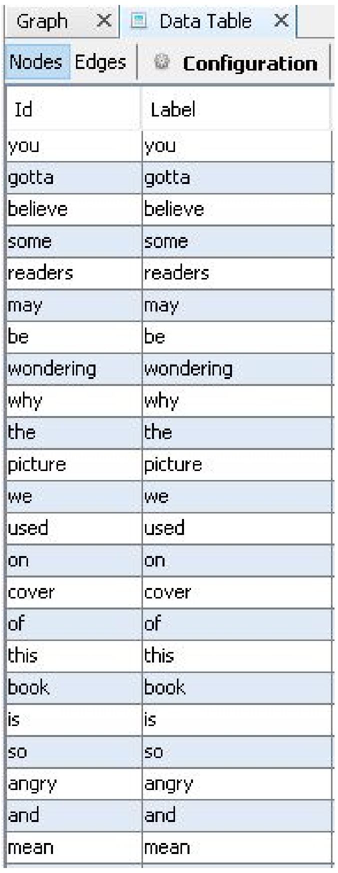
We managed to create just that single table. It would take a huge effort to create a node table without repeating *ids* associated with each word, if we consider the book has more than 25000 words.

With this strategy, we could generate two tables in *Comma-separated Values (CSV)* file format (resembling the text file format, with the same spreadsheet pattern) from that single edge table, using Gephi. These two *CSV* documents were used for further applications

## *C. Gephi as an analysis and formatting tool*

We used Gephi to generate manifold graphs according to the data extracted from the Java program. We managed to, with only a single edge tables, create a complete graph model and calculate important metrics.

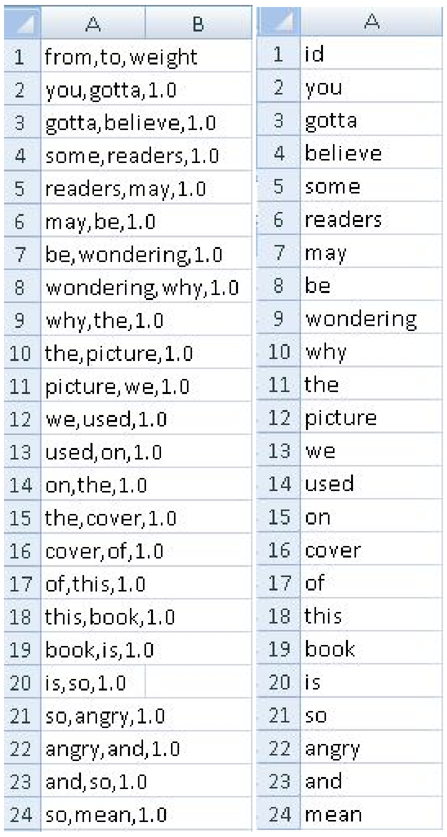
Gephi worked as an intermediate step. It automatically created a node table, with unique *ids*. In *Fig. 4*, where a part of it is shown, the *Label* column repeats the values of *Id*, as *Label* is the name that appears in the graphs, for purely visual aspects. *Ids*, in contrast, are identifiers. If two nodes have the same *ids*, then they are the same one.



| Id | Label |
|---|---|
| you | you |
| gotta | gotta |
| believe | believe |
| some | some |
| readers | readers |
| may | may |
| be | be |
| wondering | wondering |
| why | why |
| the | the |
| picture | picture |
| we | we |
| used | used |
| on | on |
| cover | cover |
| of | of |
| this | this |
| book | book |
| is | is |
| so | so |
| angry | angry |
| and | and |
| mean | mean |

*Fig. 4. A part of the node table created in Gephi*

We exported two CSV spreadsheets from Gephi. These spreadsheets were used in a program we made in R programming language, using *RStudio IDE* and *Igraph* library to analyze graphs. As *Igraph* uses a slightly different format, we manually changed the header. The results are shown on the left side of *Fig. 5.* On the right side, it is shown a piece of the final node table.



|  | A | B |
|---|---|---|
| 1 | from,to,weight | |
| 2 | you,gotta,1.0 | |
| 3 | gotta,believe,1.0 | |
| 4 | some,readers,1.0 | |
| 5 | readers,may,1.0 | |
| 6 | may,be,1.0 | |
| 7 | be,wondering,1.0 | |
| 8 | wondering,why,1.0 | |
| 9 | why,the,1.0 | |
| 10 | the,picture,1.0 | |
| 11 | picture,we,1.0 | |
| 12 | we,used,1.0 | |
| 13 | used,on,1.0 | |
| 14 | on,the,1.0 | |
| 15 | the,cover,1.0 | |
| 16 | cover,of,1.0 | |
| 17 | of,this,1.0 | |
| 18 | this,book,1.0 | |
| 19 | book,is,1.0 | |
| 20 | is,so,1.0 | |
| 21 | so,angry,1.0 | |
| 22 | angry,and,1.0 | |
| 23 | and,so,1.0 | |
| 24 | so,mean,1.0 | |

|  | A |
|---|---|
| 1 | id |
| 2 | you |
| 3 | gotta |
| 4 | believe |
| 5 | some |
| 6 | readers |
| 7 | may |
| 8 | be |
| 9 | wondering |
| 10 | why |
| 11 | the |
| 12 | picture |
| 13 | we |
| 14 | used |
| 15 | on |
| 16 | cover |
| 17 | of |
| 18 | this |
| 19 | book |
| 20 | is |
| 21 | so |
| 22 | angry |
| 23 | and |
| 24 | mean |

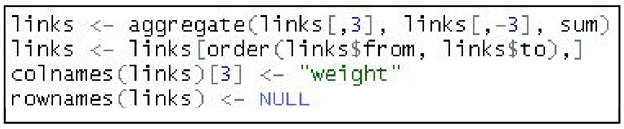
*Fig. 5. A part of the spreadsheets generated by Geph. Header of the edge table was changed manually to be used in R.*

## D. R program

With all the data finally processed and arranged in tables, that could be load in *R* programs with *I*graph library, we managed to develop and run several algorithms, having some insight over word predicting. In practice, we were able to make a program that predicts words and creates phrases according to user inputs.

Both node and edge tables were read by the program and data from them were allocated in simply two variables, *nodes* and *links* [1]. As we represented each connection between words as edges with a weight value of 1, if a word *B* is followed by word *A n* times, for example, then there would be *n* edges (A, B) with a weight value of 1. However, for graph analyze efficiency running algorithms in R, we needed a single edge for each relation between two unique words. For that, we used the code presented in *Fig. 6.*

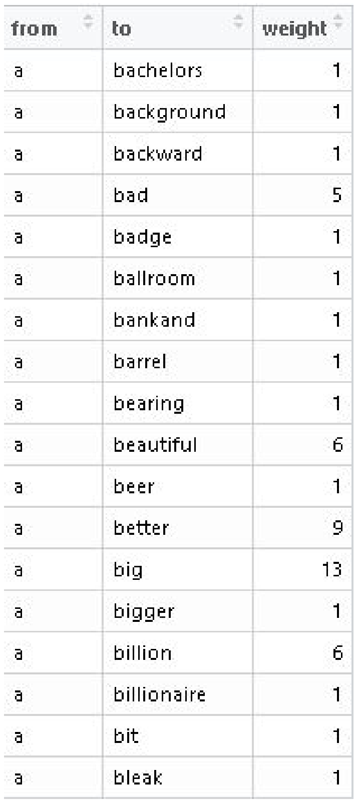
With this piece of code, we managed to aggregate all the repeated edges, summing their weights as well [1]. We also ordered the sequence edges are organized in the edge table, making the visual analyzes clearer.



```
links <- aggregate(links[,3], links[,-3], sum)
links <- links[order(links$from, links$to),]
colnames(links)[3] <- "weight"
rownames(links) <- NULL
```

*Fig. 6. Piece of code that aggregates edges of the same nature*

In *Fig 7*, all the terse and concise data extracted is displayed in a final edge table. We accessed this table in *RStudio*.

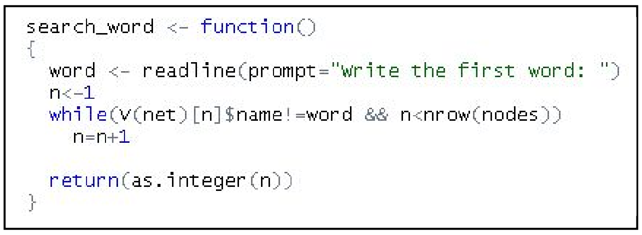


| from | to | weight |
|---|---|---|
| a | bachelors | 1 |
| a | background | 1 |
| a | backward | 1 |
| a | bad | 5 |
| a | badge | 1 |
| a | ballroom | 1 |
| a | bankand | 1 |
| a | barrel | 1 |
| a | bearing | 1 |
| a | beautiful | 6 |
| a | beer | 1 |
| a | better | 9 |
| a | big | 13 |
| a | bigger | 1 |
| a | billion | 6 |
| a | billionaire | 1 |
| a | bit | 1 |
| a | bleak | 1 |

*Fig. 5. A piece of the final edge table*

The main idea is that the users write the first word, the number of predicted words and, given that information, the program returns a phrase. For that, an important step our program need to execute, prior to word prediction, was to find the *number id* behind a specific word. We simply created a function, shown in *Fig 8,* called *search_word*.

The main behavior of this algorithm is resumed by incrementing a variable *n* until the maximum number of nodes, given by *nrow(nodes)*. We created a graph *net*, with the variables *nodes* and *links* and, therefore, we could represent each node by *V(net)[n]*. Thus, we compared each vertex name to the expected word (user input)



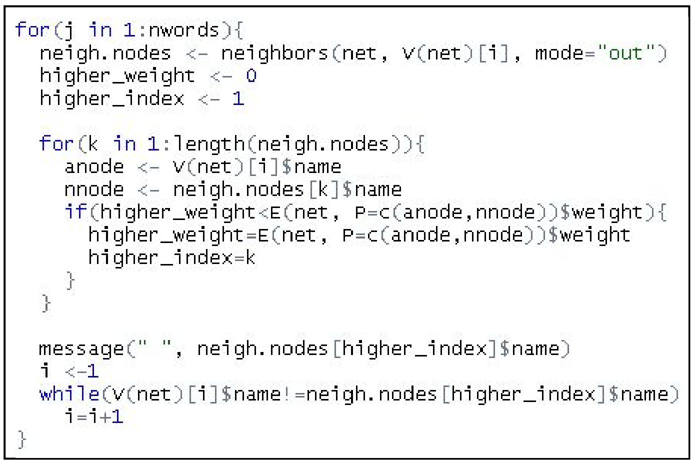
```
search_word <- function()
{
  word <- readline(prompt="write the first word: ")
  n<-1
  while(V(net)[n]$name!=word && n<nrow(nodes))
    n=n+1

  return(as.integer(n))
}
```

*Fig. 8. Algorithm that finds the number id behind a specific word*

For the prediction, we used the code in *Fig. 9*. After finding the *number id* of a word *A* the user chose, we used the *neighbors* function to find the vertices that are connected to that word *A*. The algorithm is summarized by *j* cycles, where *j* is the number of words the user wants to predict. For each *i* word, which we want to know the next one, we compare the length of all the edges formed by that word *i* and all its *k* neighbors.

The important information this piece of code returns is the index of the edge with higher weight, given by the *higher_index* variable. With this variable, the program is able to print the predicted word using the function *message*



```
for(j in 1:nwords){
  neigh.nodes <- neighbors(net, V(net)[i], mode="out")
  higher_weight <- 0
  higher_index <- 1

  for(k in 1:length(neigh.nodes)){
    anode <- V(net)[i]$name
    nnode <- neigh.nodes[k]$name
    if(higher_weight<E(net, P=c(anode,nnode))$weight){
      higher_weight=E(net, P=c(anode,nnode))$weight
      higher_index=k
    }
  }

  message(" ", neigh.nodes[higher_index]$name)
  i <-1
  while(V(net)[i]$name!=neigh.nodes[higher_index]$name)
    i=i+1
}
```

*Fig. 9. Word prediction algorithm*

## IV. Results

Using these several tools, we were able to create the desired graph and analyze it to understand better the main characteristics and patterns of the text. In this case, we could infer a little of Donald Trump's pattern of behavior and personality. This, indeed, can be done for any other text, regardless of the author.

We also performed a simple model of prediction of words. Even better, we were able to construct a whole phrase with simply two user inputs: the first word and the number of following words.

### A. *The Graph*

The obtained graph had a huge number of nodes. In order to have a better visualization, we used Gephi filters and focused in some specific intervals of degree. Thus, we could see the most recurrent words in our graph, as other characteristics.

The first step was to understand how the graph is organized, i.e, the proportion it respects and a possible power law associated. For that, we calculated the *degree distribution*. In this moment, we were not concerned with a specific *out-degree* or *in-degree* value, as it was important to understand the overall behavior. This way, we would be able to:

i) Infer the most recurrent words in Trump's speech and check if they alone synthesize the main characteristic of the text.

ii) Filter important words to use as user inputs and construct different phrases from them, in the R program

iii) Understand the program behavior and enhance the algorithm.

*Degree distribution* calculation gave us an important comprehension. Most of the nodes have a very low *degree distribution*, while very few nodes have many neighbors, as *Fig. 10* shows. This implies that there are very few keywords in Trump's speech. We, therefore, focused on that.

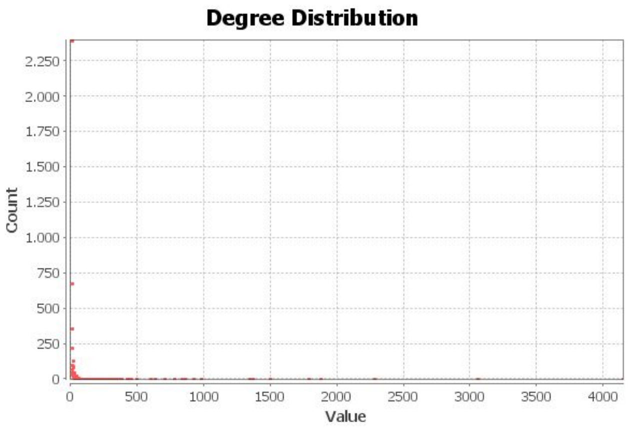


*Fig. 10. Degree Distribution of the graph*

Focusing on the most frequent words, we were able to construct a graph (*Fig. 11)* that also reflects the importance of each word in the text. The color of the node is determined by the *betweenness centrality*, i.e., how frequent a word appears in the shortest path between two other words, and the size is determined by the degree. It's possible to infer, for high degree and *betweenness centrality* values, that they are related.

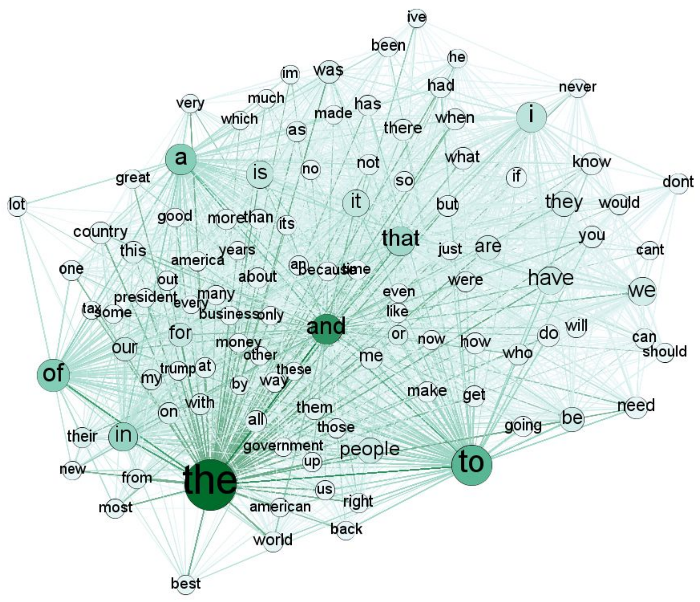


*Fig. 11. Graph of the most recurrent words. Darker green nodes are the ones with the higher betweenness centrality. Larger nodes are the ones with the higher degree value.*

As expected, the graph shows that the most repeated words are articles and connectives. Indeed, they are frequent for being an important piece of phrase construction, and they have high *betweenness centrality* for connecting words. Despite this, we can observe the recurrence of some other words, such as *people, country, government, America, money, business, me and world*. Since we had, as a database, the book of Donald J. Trump, this result suggests that the main subject that is treated by the book revolves around these words. This was taken into consideration to test the R program and will be discussed posteriorly.

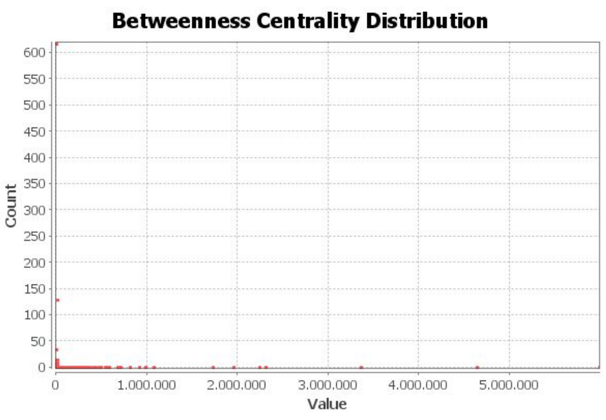


*Fig. 12. Betweenness Centrality Distribution*

The *betweenness centrality* chart is present in *Fig. 12*. Comparing this chart to the degree distribution, we may see a clear correlation

We also tried to understand how connected nodes are. For that, we managed to reconstruct the graph, changing again the color and sizes of each node. This graph (*Fig. 13)* show us every word is well connected (by the color). We also used the size to represent the *closeness centrality*, that measures the average shortest path between one node and all other neighbor nodes. Indeed, the huge node mass has almost the same size (we did not consider the border ones for they having fewer connections).

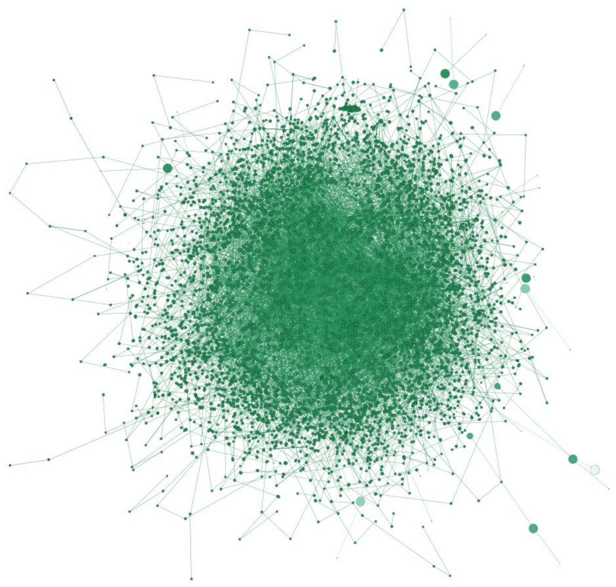


*Fig. 13. Color represents how strong  nodes are connected and size represents closeness centrality*

## B. *Reports of the graph*

We run a series of algorithms and calculated several metrics in order to identify the characteristics of the graph. They were very important to enhance the way our program in R worked, as we would be able to fix some loop issues (presented later).

Some metrics showed valuable information about the graph.  They were calculated along with the graph reports

- Average Clustering Coefficient: 0,193.
- Density: 0,002
- Diameter: 11
- Average Path length: 3.5
- Average Degree: 16,9
- Average Weighted Degree: 8,456

We observed that, although some nodes have several connections, the *density* of the graph is very small, and, consequently, there are many nodes with very few connections. Thus, many words appear few times in the book, as we could see in the chart of *degree distribution* (*Fig. 10)*. Therefore, our graph is sparse and very little clustered.

It's important to notice, also, that, although the graph has 5037 nodes, the *diameter*, i.e, the longest shortest path between two nodes, is of 11 steps. More than that, the *average path length*, in other words, the average number of steps along the shortest paths between all possible pairs of nodes, is of just 3.5, that implies everything is well connected.

Finally, taking the *average clustering coefficient* into account, we will see its value is of 0.193. Everything indicates that this graph, in overall, behaves as a *small-world network*. Comparing this value to the clustering coefficient of other known social networks, as YouTube and Flickr, or even the Web, it's clear that this clustering coefficient is of the same nature. This can be seen in *Fig 14*. The data was collected from a paper presented in the *Internet Conference Measurement*, named *Measurement and analysis of online social networks* [2].

| **Social Network** | **Clustering Coefficient** |
|---|---|
| **Web** | 0.081 |
| **Flickr** | 0.313 |
| **LiveJournal** | 0.330 |
| **Orkut** | 0.171 |
| **YouTube** | 0.136 |

*Fig. 14. Popular social networks and their clustering coefficient (Measurement and analysis of online social networks, IMC 2007)*

In fact, if we regard all the metrics calculated so far and the graph in *Fig. 13,* we may see that every word is connected by a small number of words, similar to what Milgram showed in his experiment (instead of words, he studied people). The key property of this, at last, is the *Small World Effect* [3]. Most vertices are not neighbors of each other (and we see that by looking the *clustering coefficient*), but to reach a node from any other one, it takes a few steps in average, and it's shown by the *average path length*.

To investigate that, we created a graph without a filter (*Fig. 15)*, and changed the color according to the *clustering coefficients*.

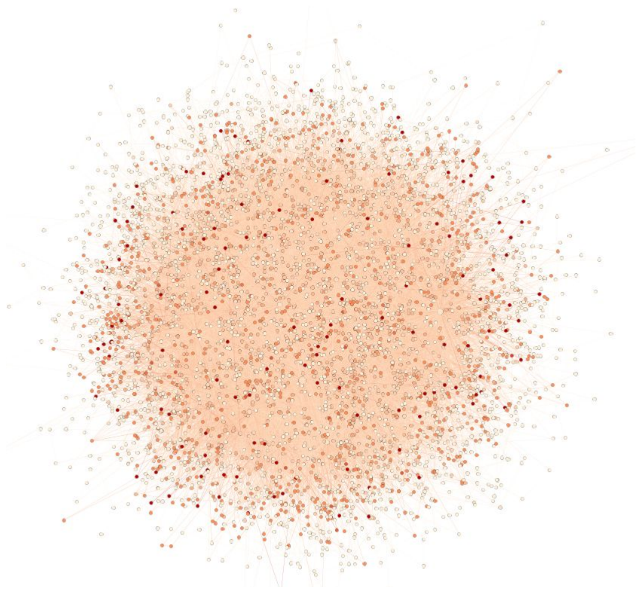
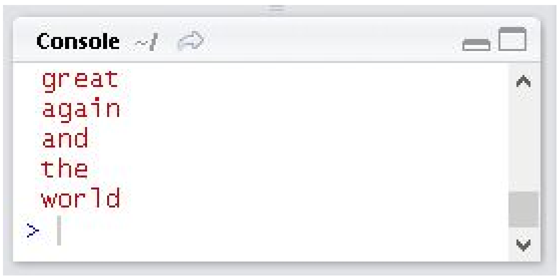


*Fig. 15. The graph without degree filters. Everything is well connect through the clustered nodes*

## C. *Predicting phrases*

Our model of phrase prediction using graph modeling is just a first insight and is based essentially on the precedent word of the sentence and not in the sentence in its totality.

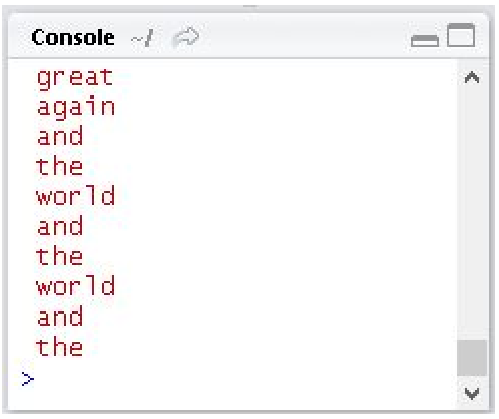
To test our R program, we chose the word *America*, that was one of the keywords observed in *Fig. 11,* and asked for 5 more words in *RStudio*. The results are presented in *Fig. 16*.



```
great
again
and
the
world
>
```

*Fig. 16. RStudio console showing the program predicting words*

This sentence refers to a phrase constantly repeated by the author, showing the efficiency of this phrase prediction model. However, we can see through the *degree distribution* that there is a considerable difference between the number of high *degree* vertices and low *degree* vertices (*Fig. 10*). As a result of this considerable discrepancy, the probability of predicting a word with high *in-degree* value is much higher than the probability of predicting a word with low *in-degree* value, which implies that, sooner or later, the program will reach that word of high *in-degree* value again and fall into a loop after three or four words. If we choose the word *America* one more time and ask for 10 more words, we will have the printed message in the console shown in *Fig. 17*as result.



```
Console ~/
great
again
and
the
world
and
the
world
and
the
>
```

*Fig. 17. RStudio console showing the program predicting words in a loop*

## V. Conclusions

The use of graphs and their properties is very useful for describing how things are connected and how they form networks. With our simple program, we could have an insight on how useful tools, such as the one of phrase prediction developed by Google, can be created with just simple concepts of communications and network.

We were able to classify our network, recognize its *small world effect*, and identify the reason why it's very well connected (the connectives and articles). Thus, we could consider only the most recurrent words and see the results. We showed results for only the word *America*, but it works to any other present in the textbook. Future enhancement will be provided for this work: the next goal is to improve the efficiency of word search and make it run as fast as applications available in smartphones nowadays

### A. *The graph*

We observed that, although some nodes have several connections, the *density* of the graph is very small. It suggests that there are many nodes with very few connections, i.e. many words appear few times in the book, as we can see in *Fig. 10*. Therefore, our graph is sparse and very little clustered.

As it is very little clustered, most vertices are not neighbors of each other. However, the average path length is small. This indicates that to reach a node from any other one, it takes a few steps, and confirms the behavior as a *small-world network.*

The explanation behind that can be provided in the next section, the *frequency of words.*

### B. *The frequency of words*

After filtering the obtained graph, if we decrease it slowly, we will be able to see that the *top 5* most recurrent words (*the, to, and, of, a)*. They have much greater *degree* value than the other words and also have higher *betweenness centrality*. These five words are connectives and articles, which is expected in a text.

If we decrease a little more the filter, we will observe the recurrence of some pronouns and, finally, some nouns, which is more relevant to our study because we can relate this class of words with an individual's vocabulary. The more recurrent nouns were: *people, country, America, money, government, president, world, Trump and business.* These results suggests that the main subject that is treated in the book revolves around these words, and these composes the more often vocabulary of this individual in this book.

The high occurrence of articles is also due to the way in which our program was constructed. If we analyze line by line the text, we will notice the program disregards periods and, because of that, the article that starts the next sentence appears as subsequent word of the last word of the previous sentence. However, this was part of our abstraction, since our objective was to have an insight over prediction.

### C. *Phrase Prediction*

With our simple program we could have an insight on how useful tools can be developed with just simple concepts of communications and network. The program works well with the available tools, although it does not present the same efficiency as actual smartphone applications. Ths was expected, as theses applications have been improving constantly, for having a considerable quantity of users and developers working on it.

Graphs should be considered in these softwares. It may enhance complement the behavior of such apps and give a better feedback on how the sentence is being constructed.

## VI. References

[1] K. Ognyanova. "Network Analysis and Visualization with R and igraph", Wroclaw, Poland, 2016

[2] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, B. Bhattacharjee, "Measurement and Analysis of Online Social Networks", IMC, 2007

[3] S. Milgram, S. "The small world problem". Psychol. Today 2, 60–67 (1967)