Analysis of the STCP transport network

Diogo Bogas

up201202482@fc.up.pt
Departamento de Ciências de Computadores
Faculdade de Ciências da Universidade do Porto

January 29, 2017

Abstract

Abstract

This article is the final product of a study on a transport network that operates in the city of Porto, as a complex network.
I started by creating a way to extract all the information considered usefull automatically. Then upon deciding on how to store it locally, more data was derived about the network. Some of this data was derived as an input for gephi, some was generated by it.

Keywords: transport network, complex networks

1 Introduction

In the real world, lots of information can be represented by complex networks. Some examples are chemical systems, neural networks, the Internet and the World Wide Web[**?**]. The following two concepts are related with the data gathering phase of this study.
The first one is XMLHttpRequests. The XMLHttpRequest is an interface that allows scripts to perform HTTP client functionality, such as submitting form data or loading data from a remote Web site[**?**]. The next concept that the reader should be aware of is JSON. JSON stands for JavaScript Object Notation, and is a lightweight, text-based,language-independent data interchange format[**?**]. Knowing this, we introduce the network that this article focuses on. In Porto, the most influential public transport company is STCP. The bus network is large enough to cover the entire city and some peripheral areas that are part of the district.
To be able to say anything about this network, first and foremost, i started searching on where to collect the data from. Ideally, some sort of database was optimal, but the most reliable place to get information ends up beeing the site, as it has lots of information usefull to this study, and it is updated regularly, as there can be alterations to the network, justifiable with construction works in some street used by it. After having solid information, the analysys done consists mainly on degree distribution and charateristic patterns about the gathered information, and on facts derived from it. This article begins by exposing some relevant concepts to understand exactly how the information was colleted, and why it is reliable.
Then, in a separate section, it begins by explaining said gathering, continues by solving the storage problem and finally describes what type of process-

ing was done, and to what end.
Then, in the Results section, all the analisys done on the network is shown, along with the interpretation of the results.

2 Data Gathering and Processing

2.1 Data source

The first decision made in this study wasn't what to study about the network, was to know were to get the data.
The website of the company has a dropdown box that allows us to pick a random line, and shows us all the stops it goes through, and a map made with a google maps API.
At first, webscraping was the route followed. Web scraping [**?**] is the set of techniques used to automatically get some information from a website instead of manually copying it.

Altough this seemed to solve the problem, i found that it wasn't efficient, as i had to ,manually, iterate through the options presented and provide the HTML as input to a piece of code. Not beeing automatic , it would be painfull to update our information at any given time. The best way to do it, is to read the answers the server gives the client when an option in the dropbox is selected. Luckily, this answer is in form of JSON objects, which can be processed with some ease.
The first answer i took a look at, had a list of all the options on the dropdown box. This meant all the lines this company has operating were there.
Something important to refer, is that each line has two directions. For example: one of the lines, line 207, circulates between two stops, Campanhã and Mercado da Foz. It can start in Campanhã and go to Mercado da Foz and vice versa. While doing so, it may or may not go through the same stops, as there are one way streets in Porto that the line may use. To have accurate data, 207 by itself can't be a bus line. However, 207 in direction 0 can, as 207 in direction 1.
The next answer had an array of all the stops that line went through. At this point, iterating through all the lines, we could know all the stops in the network.
While looking at a line, if we click on a stop, we are redirected to a page that describes it. The same technique we used for the lines, we can use here.

The parsing of the answers the server sends the site is fully automated, using the JAVA programming language, folowing the next line of thought:

1. read the answer that gives all the lines
2. for each line : store it locally and collect all the stops
3. for each stop colected : get the details and store them

About the tecniques used, it is still needed to mention the reliability of each one, and why the technique used is far superior.
The method used follows a 3 steps line of thought that gives us every piece of information about the network, is automatic and takes at most five minutes.
Webscraping would follow the same algorithm, although it would envolve manual input of data, which would take much more than five minutes, and is prone to errors, as there are almost 130 different lines and close to 2500 stops.

The information the site gives about bus stops and bus lines is enough for the common user, but we need more. There are some aspects we need to derive if we want to do a complete study on the proposed network. Therefore, every piece of information was locally stored.

2.2 Data storing

While reading the information from the server answers, i noticed that each time i wanted to refresh my data, the running time of the script in charge of that was approximately 5 to 10 minutes. To test some methods that depended on that input, that wasn't viable. So, a simple answer was to create two .txt files with the information gathered.

One of the files (AllLines.txt) has all the information about each bus line. Each bus line is represented in a single line by a code, a direction, an integer that tells us the total number of stops in that line, and finally, all the stops that compose said line.

The other file (AllStops.txt) has all the information about all the stops. Each stop is represented in a single line by a stop code, an address, a zone, a name and a pair of floating point numbers that represent it's geographical location.

All of this information can be refreshed, as there are methods that do so.

There is the need for some structures in which to represent all this data we have, so it can be easily used to answer some of the questions raised above. The first type of information stored was information about lines, and the structure to represent a line is called BusLine:

- Direction
- Accessibility
- Code
- Description
- Pubcode
- LineStops

Direction is an integer that can take the values 0 or 1 and represents the direction a bus goes through a line.
Accessibility is an integer that dictates the what special access conditions a bus line has. It can have wheel chair access, baby stroller access for example. Its a variable that was created when the information was being gathered, but ended up remaining in the structure.
Code is the server way to tell bus lines from each other.
Description is a String that tells us what appears in front of a bus that if going through a line.
PubCode is the people's way to tell lines from one another.
And LineStops is a List of Strings that represents the Stops that compose a line.

After knowing each line's composing stops, a list of all the stops was constructed, and the info for each stop was stored locally. The JAVA structure used is called Stop:

- StopCode, the ID of a Stop.
- Address.
- Zone, the part of the city a Stop belongs to.
- Name, the way people know which stop is which.
- Longitude.
- Latitude.

Thinking of this network as a graph, we already have the nodes in form of stops. Now we need to derive the edges. The edges are derived from the lines, taking into account the order each line goes through the stops it serves.The Edge structure used here is as follows:

- source;
- target;
- desc;
- weight;

Source and Target define the edge, as the network is a directed graph. Desc is a String composed by both source and target stopcodes and the weight variable represents the number of lines that use that edge.

To make this a more complete study, the stops were grouped by 2 characteristics, the first being their address. In this case, each street was a node, and an edge represents a trip between the streets of two stops. The node is represented by the BusStreet structure:

- street
- stops, a list of all the stops in that street
- longitude;
- latitude;

The edges in this type of grouping are represented by AdressEdge:

- src
- dest
- weight
- nome

The graph is directed still, weight is the number of lines that use that particular interaction and nome is a way to differentiate the edges, and is derived from the source and target streets.

The final type of grouping was by code prefix, which is easier to explain by an example:
In the AllStops.txt file there are two stops which stopcodes are TSL1 and TSL2. These were grouped in a single object that is named TSL. The structure to store this type of node is called Spot:

- code
- stops
- LinesServed

Code is the way to tell Spots apart,stops is a list of stops the Spot has in itself and LinesServed is a list with all the lines that go through that Spot. The edges here are called SpotEdge:

- from
- to
- weight
- name

"from" and "to" are the source and target Spots, weight and name, as the other types of edges are the number of lines that use the edge and a way to distinguish the edges respectively.
Some files, six to be more precise, were not included in this subsection, as they were generated with JAVA to be used by gephi.These files are described in the next subsection as the respective generating methods.

2.3 Data processing and input creation

As mentioned before, Gephi was used to study the network. Gephi is a pretty powerful tool, but it requires some form of input. In this particular case .csv files. This subsection describes the methods that process the data (stored in any way described in the previous subsection) and generate the input Gephi needs.
Still on the previous subsection, it was mentioned that there were 6 files left to describe. Three of these are groups of nodes, and the other three, groups of edges.

The first file to be described is called stops.csv , and it represents every single stop there is in this network. Five columns per stop :

- Id , is the code of each stop (the unique attribute).
- Address
- Zone
- name
- latitude
- longitude

To make this file, the method makeNodesCSV was created. It begins by printing "Id;Address;Zone;Name;Longitude;Latitude" in the first line of the output file. Then, for each stop it reads from the allStops.txt file, prints its attributes by the order sugested in the first line. As in the source file, each stop only takes one line.

The second file to talk about is called allEdges.csv, with four columns per edge:

- Source stop
- Target stop
- type which is going to be directed for every edge in this study
- weight

The generating method to this file is allEdgesCSV, and is similar to the nodes one. It also begins by printing what each column represents, (in this case "Source;Target;Type;Weight") and then for each edge it receives, prints its attributes in the order required.
Because the stops were also grouped by address, we need two more files for that, the node file for that grouping is called streetNodes:

- Id
- totalStops
- longitude
- latitude

This first file is generated by the makeStreetNodeCSV method, which in the first line of the file writes "Id;TotalStops;Longitude;Latitude", and in each subsequent line writes each streetNode's values.
And the edges file is called streetEdges:

- weight
- source
- target
- type

The second file for this type of grouping is generated by makeStreetEdgesCSV, and this file holds, for each edge, the weight, source, target and type.The weight in this particular case represents how many bus lines use that particular edge.

The last type of grouping is by code prefix. This allows us to group spots that are near each other (and potentially have a similar name) in a single structure we call Spot:

- code
- stops
- LinesServed

Code is the way to tell Spots apart,stops is a list of stops the Spot has in itself and LinesServed is a list with all the lines that go through that Spot. The code variable derivation is easy to exemplify: in our list of Stops there are two whose codes are TSL1 and TSL2. These were grouped in a Spot which code is TSL.

The edges in this type of grouping are represented by the SpotEdge structure:

- from
- to
- weight
- name

From and to are the source and target Spots, weight and name, as the other types of edges are the number of lines that use the edge and a way to distinguish the edges respectively.

3 Results

To better understand the results obtained, i'm introducing the final two concepts. The first one is closeness centrality. Closeness centrality measures how many times a cenrtain node acts as a bridge on the shortest path between two other nodes. All the centralities measured were normalized to values between 0 and 1. The last concept is excentricity. This represents the longest shortest path that can be measured starting at a certain node. This first image is the very first piece of information that our tool gave us, a visual representation of all 2416 stops in the network(not using any type of grouping), colored by zone. All of these stops interact between them a total of 2823 times to create 142 lines. Using the GeoLayout i got:

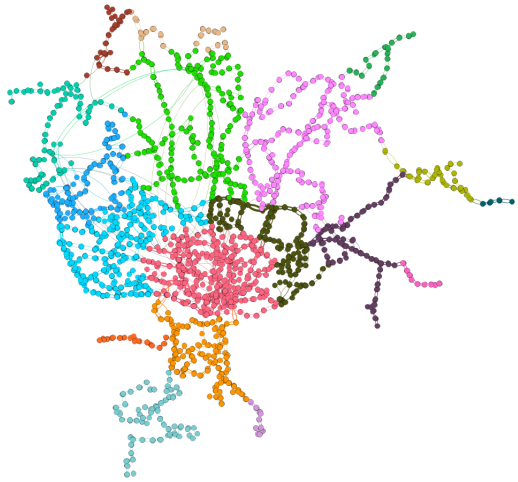

Figure 1: The network, without grouping, with stops colored by zones and shown according to geographical coordinates

The following image is the palette used to color the network shown above, and it ends up giving us how large is each zone relatively to it.

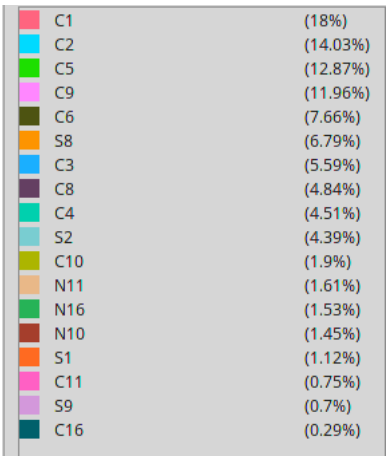

Figure 2: A table with all the zones, sorted according to size,displaying each zone's color

Here is displayed the top 10 in terms of excentricity:

Stop	Excentricity
Rio Tinto (Estação)	136.0
Cabine	135.0
Perlinhas	135.0
José Poças	134.0
José Portugal	134.0
Lourinha	134.0
Cedofeita	133.0
Monte	133.0
Piscinas	133.0
Bicheiros	132.0

The second question raised in the Introduction section of this report was "Which Stop has the most lines going through it and how many times?". For this, it's better to group the stops by code, and answer a different question : "Which Spot has the most lines going through it?", as a Spot is much more influential than a simple Stop. Te answer to this, is in the table below,extracted from spotNodes.csv were the 5 most populated Spots are represented:

Id	totalStops	linesServed
TRD	6	21
BCM	5	20
BS	8	18
AAL	6	17
CMO	4	17

Checking the Id's in the AllStops.txt file, we get, by ascending order: Carmo,Aliados,Bom Sucesso,Casa da Música in Boavista and Trindade topping the list.Keep in mind that line 200 in direction 0 and 200 in direction 1 are two differetn lines.

In terms of streets, we want to know what is the street that holds the most stops. Checking the streetNodes.csv file and sorting by descending order the totalStops column, its verified that Estrada da circunvalação holds 86 stops. After some fact checking, it was verified that this street is 17 km long[2], which justifies a 64 stops difference to the 2nd street in this ranking. Follows the top 5:

Street	Total Stops
ESTR.CIRCUNVALAÇÃO	86
R.D.AFONSO HENRIQUES	32
AV.BOAVISTA	30
R.S.VICENTE	23
R.COSTA CABRAL	22

Analising the nodes, with help from Gephi, we now look to the betweeness centrality. Betweeness measures how many times a given stop acts as a bridge in the shortest path between two other nodes. The next table shows, the "top 10" relative to betweeness:

Nome	Betweeness
Trindade-C1	1145449.881
Graciosa-C1	771297.511
Bv.Cemitério-C1	754319.608
Moreira Sá-C1	736031.847
Asprela-C6	678387.028
Av.Aliados-C1	644478.771
Trindade-C1	612273.626
IPO(Circunval.)-C6	611095.614
Areosa-C6	601167.850
Casa da Música-C1	583327.933

There is a very noticeable pattern here, most of the Stops shown above, belong to C1 zone, the center of the network.

4 Conclusion

The results that we obtained colide with the reality.It was expected that the centre of Porto was the most busy zone of the entire network, and proof of that, is that C1 is the biggest zone,and most of it's Stops have the biggest Betweeness in the entire graph. The prior knowledge i had about the service this company provides, and the city itself, helped me as a "sanity check" as the data was processed. This study also allowed me to get a deeper understanding of Graph theory , and it's applications in real life situations.