Introduction to Computational Social Science assignment 4:

Closing triangles in a literary social network: Shakespeare - Caesar

Mihaly Hanics, MS in Social Data Science

This report is about connecting the characters in Shakespeare's tragedy of Julius Caesar based on their names being contained together in a sliding word window, with window size of T=1000, creating a social network and analyzing the triadic closures in the network. Network analysis of characters appearing in stories has been studied by Sandra D. Prado et al [1], particularly in the pieces *Alice's Adventures in Wonderland* and *La Chanson de Roland*, however, building a temporal network instead of a dynamic one. Among building a temporal network, sentiment analysis was done in the paper of Semi Min et al [2], on the novel of Victor Hugo, *Les Misérables*. In this work, the network constructed is a dynamic network.

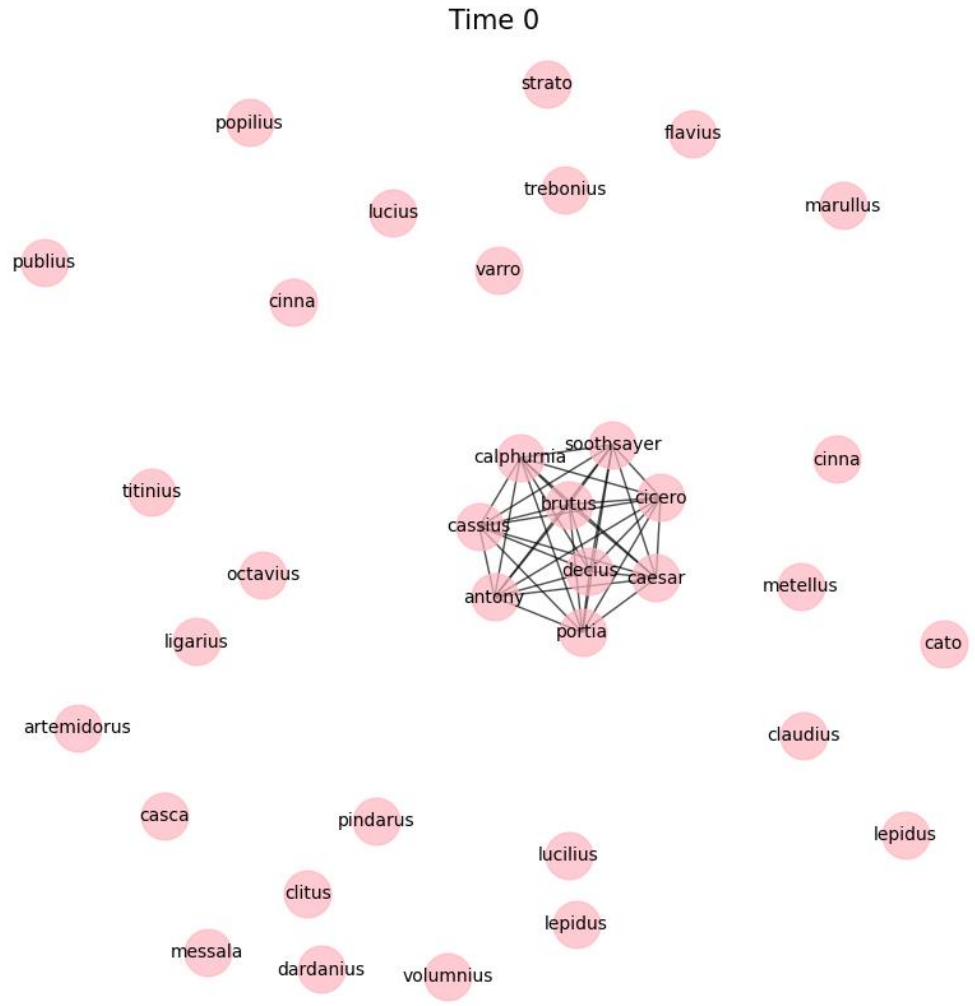
Technical details: I used the Project Gutenberg library to access the book's digitalized format, imported from the Python Natural Language Toolkit (NLTK)'s corpora. LLMs: I used GitHub Copilot for code completion. When I don't prompt it, as I'm writing code, it has the ability to continue it, for example, this snippet was written by Copilot:

```
for i in range(0, len(tokens_B), block_size):
    block = tokens_B[i:i+block_size]
    blocks.append(block)
```

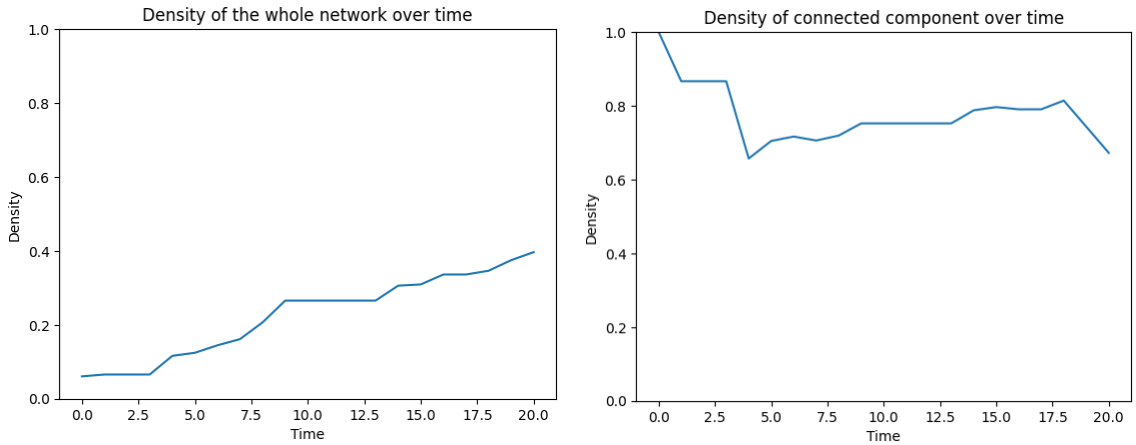
However, this is just "trivial" code completion to save time and not for coming up with ideas to tackle the problem. Commonly it just completed some code where I did something similar as before. I prompted Copilot Chat with "/fix convert igraph code to networkx" (when I mistakenly treated a NetworkX graph as an igraph graph), to which (without answer, just showing a possible solution) suggested code that did indeed which Copilot did help a lot in writing the code faster, and also probably I'd make more mistakes without these code completions. I also was indirectly helped by ChatGPT: a student gave me code for visualizing correctly in NetworkX, which I copied and modified.

Preprocessing: I manually added the names of the main characters in the text. The window size was chosen to 100 so I split the words into blocks of 100. The non-alphanumeric words were removed, and words were lowercased and tokenized.

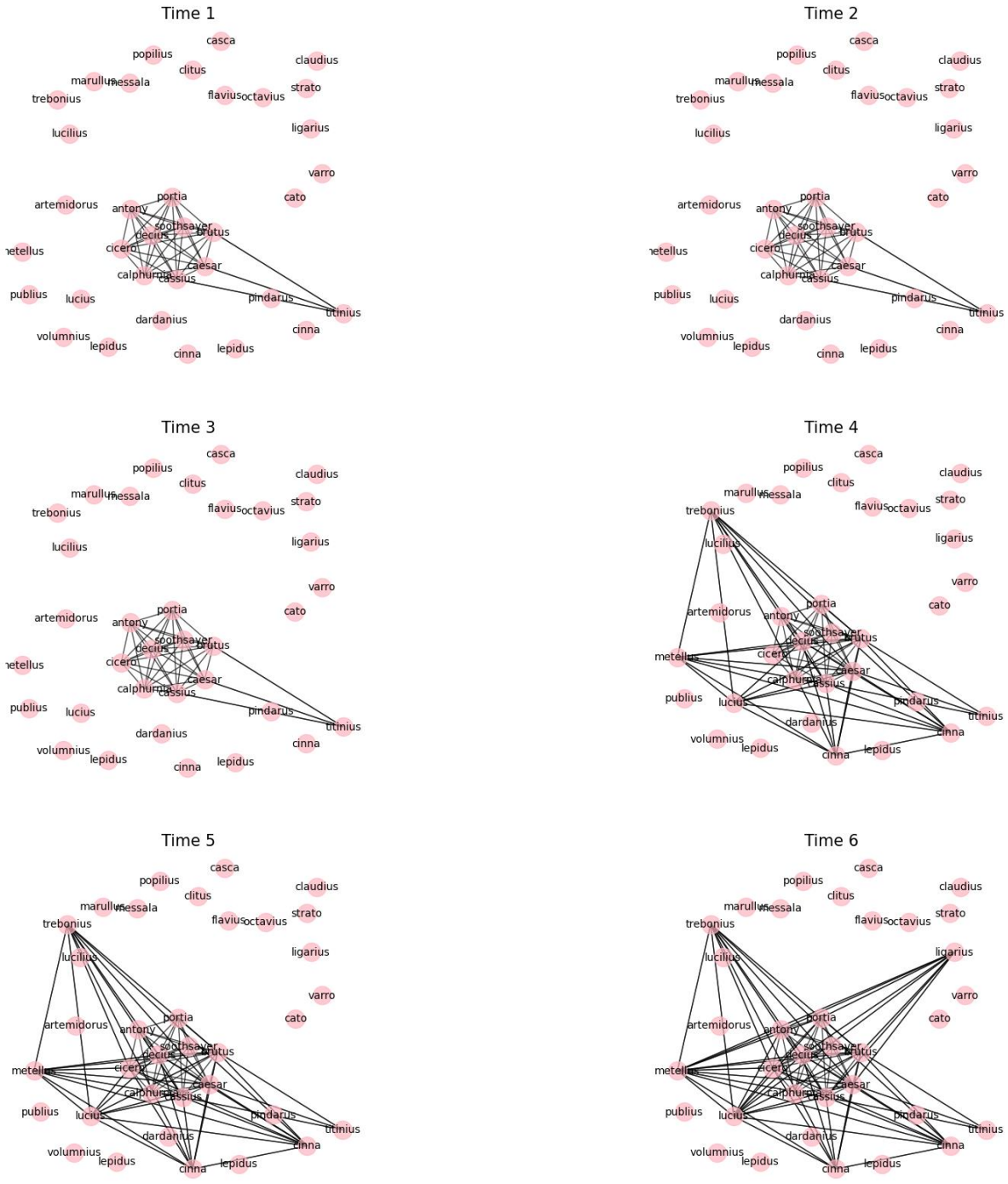
Process: To create a network, first I created a graph (in igraph, later transformed to NetworkX) of all the characters collected, and fit the word window on the first 100 words. In each step, the characters in the current word window are exported, and in the graph the corresponding nodes are connected (if not connected yet). This way, we get a dynamic network that develops overtime, and we can analyze certain properties, for example, how long it takes typically for an open triangle to become a closed one. Let's see the network after the first step:



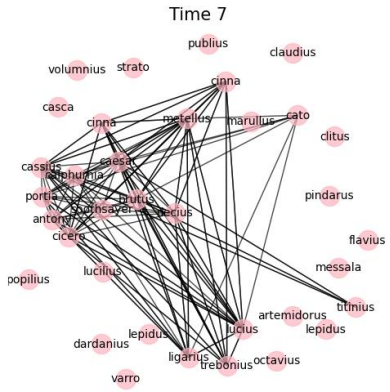
Already we can see one observation, that the “sub-initial” network’s only connected component is a complete graph, of course this makes sense, as all characters were connected with each other in step. However, as time advances, the component gains new connections and becomes non-complete, the whole network becoming more complete and denser, but the connected component over becomes less dense.

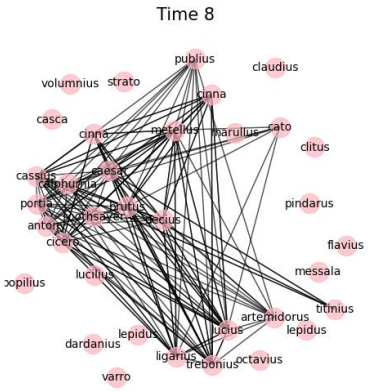


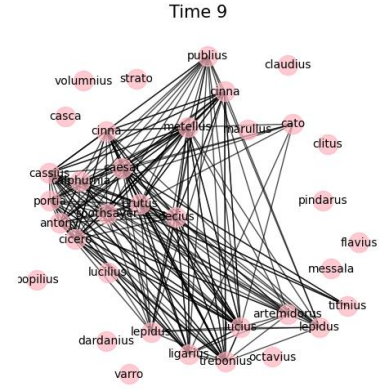
There is some slow increase / stagnation between the 5th and 19th discrete time period on the density of the connected component, but the drops by introducing new characters to the story results in major drops. Now let's see the development of the network:

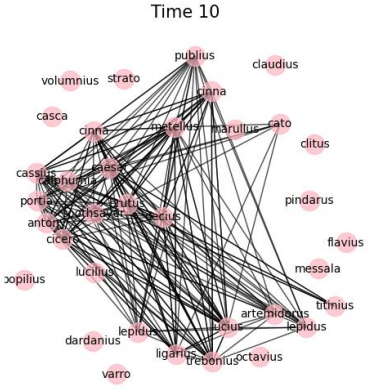


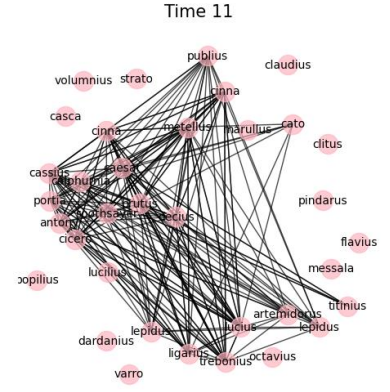
As you can see, at time 1, one more character, Titinius appears in the story, then there is no development in the next 2 steps. In the story, this makes sense, as all characters appeared in the first act appeared in the first window. The black sheep Titinius is not present in this act, his name was mentioned by Cassius. Later, Casca would have been added, however in the tokenization his name was filtered, therefore him and other filtered out characters stay as disconnected points in the graph throughout the process.

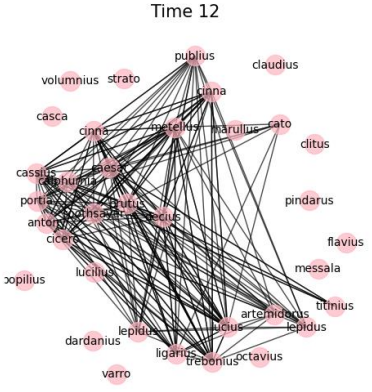

Time 7
publius
claudius
volumnius
strato
cinna
casca
cinna
metellus
marullus
cato
clitus
cassius
caesar
calphurnia
portia
brutus
pindarus
antony
cicero
flavius
lucilius
messala
popilius
titinius
artemidorus
lucius
lepidus
dardanius
lepidus
ligarius
trebonius
octavius
varro


Time 8
publius
claudius
volumnius
strato
cinna
casca
cinna
metellus
marullus
cato
clitus
cassius
caesar
calphurnia
portia
brutus
pindarus
antony
cicero
flavius
lucilius
messala
popilius
titinius
artemidorus
lucius
lepidus
dardanius
lepidus
ligarius
trebonius
octavius
varro

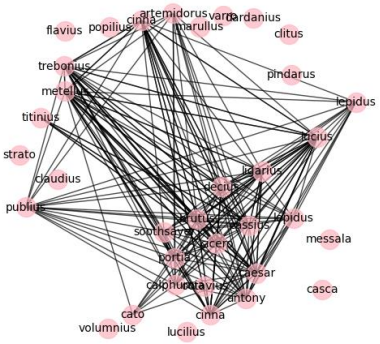

Time 9
publius
claudius
volumnius
strato
cinna
casca
cinna
metellus
marullus
cato
clitus
cassius
caesar
calphurnia
portia
brutus
pindarus
antony
cicero
flavius
lucilius
messala
popilius
titinius
artemidorus
lucius
lepidus
dardanius
lepidus
ligarius
trebonius
octavius
varro


Time 10
publius
claudius
volumnius
strato
cinna
casca
cinna
metellus
marullus
cato
clitus
cassius
caesar
calphurnia
portia
brutus
pindarus
antony
cicero
flavius
lucilius
messala
popilius
titinius
artemidorus
lucius
lepidus
dardanius
lepidus
ligarius
trebonius
octavius
varro

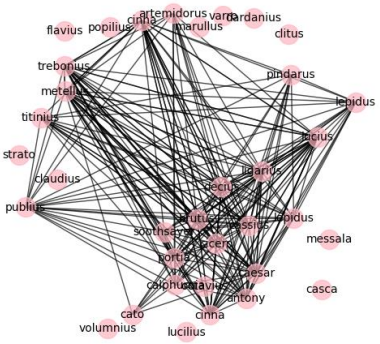

Time 11
publius
claudius
volumnius
strato
cinna
casca
cinna
metellus
marullus
cato
clitus
cassius
caesar
calphurnia
portia
brutus
pindarus
antony
cicero
flavius
lucilius
messala
popilius
titinius
artemidorus
lucius
lepidus
dardanius
lepidus
ligarius
trebonius
octavius
varro


Time 12
publius
claudius
volumnius
strato
cinna
casca
cinna
metellus
marullus
cato
clitus
cassius
caesar
calphurnia
portia
brutus
pindarus
antony
cicero
flavius
lucilius
messala
popilius
titinius
artemidorus
lucius
lepidus
dardanius
lepidus
ligarius
trebonius
octavius
varro

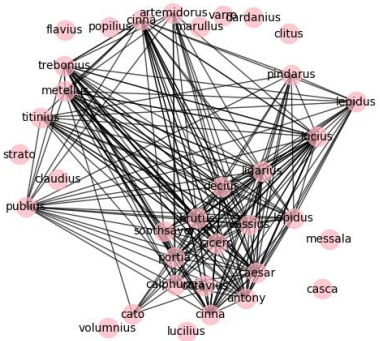
Time 13



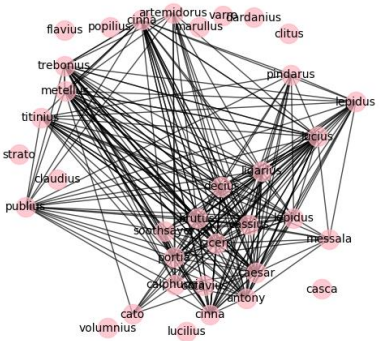
Time 14



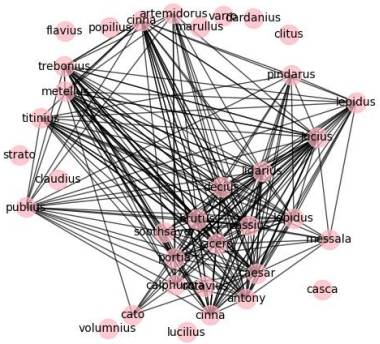
Time 15



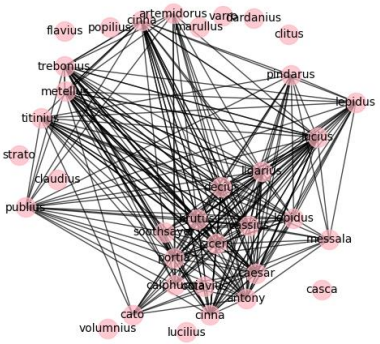
Time 16



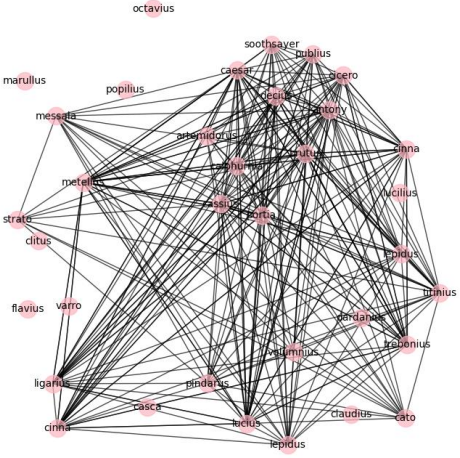
Time 17



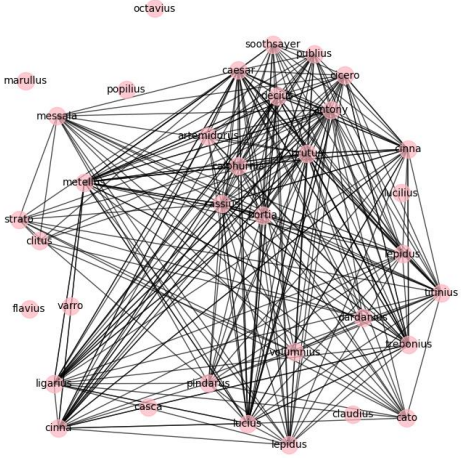
Time 18



Time 19

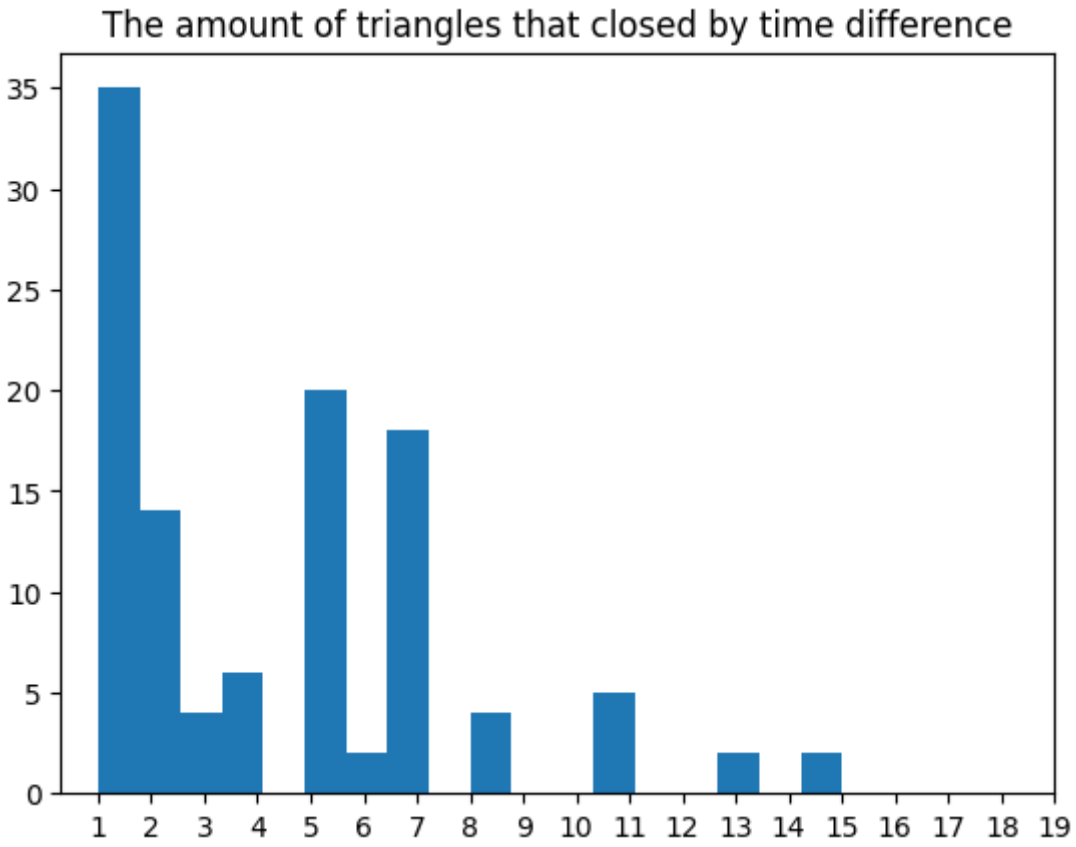


Time 20



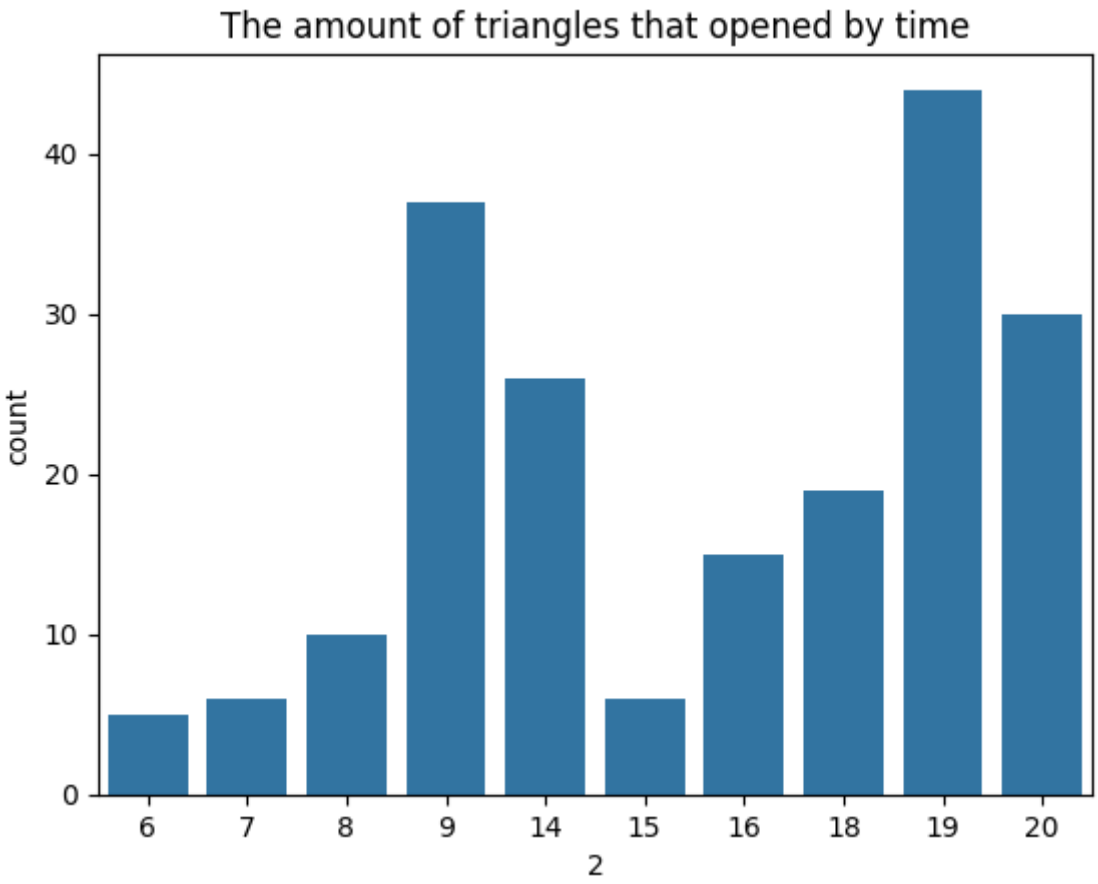
The plots shows the change of density.

We can see that in terms of triangles, the number of closed triangles compared to open triangles is a very good ratio. To explain why this happens, everytime a word block of $n+k$ characters is processed,where n are the number nodes already in the connected component and k are the number of new additions, then we get at least $\binom{n+k}{3} - \binom{n}{3}$ new closed triangles (I did not look this up, I can provide proof of the formula if needed). The maximum amount of opened triangles in a step can be $knm + k\binom{m}{2} + m\binom{k}{2}$, but this is often multiple times more than the actual opened triangles. In a totally random network, the ratio of the amount of closed triangles and open triangles is theoretically 1:3, because of the number of permutations. In the final network, we have 198 open and 112 closed triangles, which is around the ratio of 1:1.768, much more than in a random network, proving that indeed this way of constructing links breeds many closed triangles. Let's see how long it typically took for an open triangle to be closed:



In most cases, they were closed in one step, and commonly in 5 or 7 steps, however, most of those come from one time interval only. Big leaps are very rare as most characters are introduced already by that time.

It is also worthy to look at left open triangles:



Many triangles were opened in the last two blocks, as the last characters were introduced.