Assignment: Graph Analytics

Advanced Analytics in a Big Data World
Prof. dr. Seppe vanden Broucke

Contents

Dataset Description

For this assignment, you will work with a graph extracted from Twitch, the popular streaming platform (mainly for games, but also other categories such as food, travel, etc.).

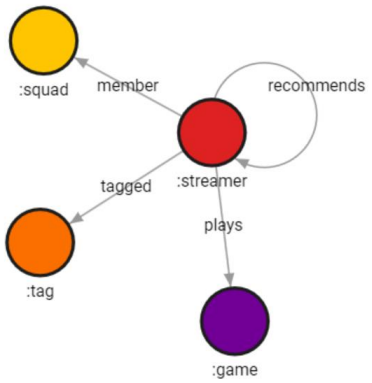
A data set has been collected for you during the period of December 2022 – April 2023.

To create the data set, the following approach was followed:

- We obtain a list of "semi-official" tags from Twitch
- We periodically scrape the front page of Twitch streamers ordered by most viewers to lowest viewers
- We visit each streamer and extract their details, including the game they're playing, tags they have placed on their profile, "squad" (group) they belong to
- We also fetch the "viewers who watch x also watch…" recommendations and add those streamers
- At the end of April, a final pass was performed based on number of times recommended to expand the graph in terms of less-popular streamers
- Finally, for all streamers, their basic details were refreshed (description and number of followers)

Note that this is by no means a scientific approach. E.g. we report statistics in terms of "number of times seen", "average number of viewers", but this is based on the scraper having visited and seen the scraper.

The resulting graph is saved in a Memgraph database. The graph schema (metagraph) looks as follows:



- General remarks:
    - All nodes have an “id” attribute which can be used as a unique identifier. This is different from the built-in “<id>” (Neo4j’s Node id)
    - The edges do not have attributes
- streamer:
    - Represents a streamer (a Twitch account)
    - id, name: unique account name
    - description: description as of end of April, if empty string, the streamer did not set a custom description, if NULL, the streamer was blocked / removed as of end of April
    - followers: number of followers, if NULL (=/= 0), the streamer was blocked / removed as of end of April
    - nr_streams: number of times the scraper saw this streamer as being live. If 0, the scraper saw this streamer being recommended by another streamer or on the front page, but never saw it live
    - views_min, views_avg, views_max: min, avg, and max number of viewers watching the streamer based on number observed when visiting the page, NULL in case nr_streams is 0
    - first_seen, last_seen: first and last time the scraper saw this streamer live, NULL in case the streamer was never seen live
- tag:
    - Represents a tag
    - id, name: unique tag name
    - letter: if the tag is present in Twitch’ tag directory, it used to get a short label assigned to it (https://www.twitch.tv/directory/all/tags/). During the past month, however, Twitch has removed this listing, though they’re still in the database. In case this is NULL, it is a “non-official” tag added by the streamer themselves (though note that the distinction between official and non-official seems to be gone)
    - description: tag description, again for “official” tags only
    - url: used to link to streamers streaming with this tag, not used any longer, the new url is `https://www.twitch.tv/directory/all/tags/<name>`
- squad:
    - Represents a squad, streamers can organize themselves in squads (teams, groups)
    - id, name: unique squad name
    - url: link to the squad page
- game:
    - Represents a game
    - id, name: unique game name
    - url: link to the overview page of streamers playing this game

Assignment

Your task for this assignment is to use Cypher queries and Gephi to analyze this data set.

You are free to explore anything you deem interesting and present your findings in your report. The main goal is to get familiar with Cypher and Gephi, but also to hone your "storytelling" skills. In that sense, try to focus on a single or a few hypotheses or findings you explore in full (with nicely formatted visualizations) and explaining what it says instead of just going for quick filter saying: "here are the three nodes with the most connections" (boring).

Below are some suggestions/tips of what you can go for:

- Perform an analysis of the most popular streamers / games / tags over time
- Perform a community mining analysis based on common tags between streamers and / or based on recommendations
- Do blocked streamers typically play particular games or use particular tags?
- Can you reverse engineer how Twitch' recommender system works? Is it based on tags, games played, number of followers?
- Can you do something with the description field of the streamers?
- (Optional) you can also use external data from e.g. https://www.igdb.com/ (this is the game database Twitch uses) – they have an API – to bring in additional game information
- (Optional) use e.g. `networkx` or one of Memgraph's MAGE algorithms (https://memgraph.com/docs/mage/algorithms) to perform some more intricate analysis (e.g. you will have access to more community detection techniques that way)

Don't try to go for all of these. Pick something you want to explore and try to work this out in full. You will realize that the data set is likely too large to look at everything at once, so a guided "deep dive" will work better. Also, trying to visualize the whole graph will most likely not lead to a lot of real insights. Focus on a "focused analysis with good insights" instead of "big data and discovering nothing".

Setting up Memgraph and Gephi

We used to work with Neo4j for this assignment, which is a solid tool supporting lots of (data science) extensions, but sadly has become so commercial lately that it is a hassle to download. It can also be quite slow. Instead, this year, we'll be working with Memgraph, which is also "open source but also commercial", but with (hopefully) the advantage that it is faster and easier to set up than Neo4j. It is mostly compatible (e.g. Cypher works the same), but some aspects (such as exporting data) are a bit trickier, so make sure to follow the instructions below carefully.

First, Memgraph only provides a container to run instead of a typical installer. Since this is becoming an increasingly common pattern, we can also take this opportunity to get (a little bit) familiarized with Docker. Download and install Docker Desktop for your platform:

- https://www.docker.com/products/docker-desktop/

Next, download and install Gephi for your OS:

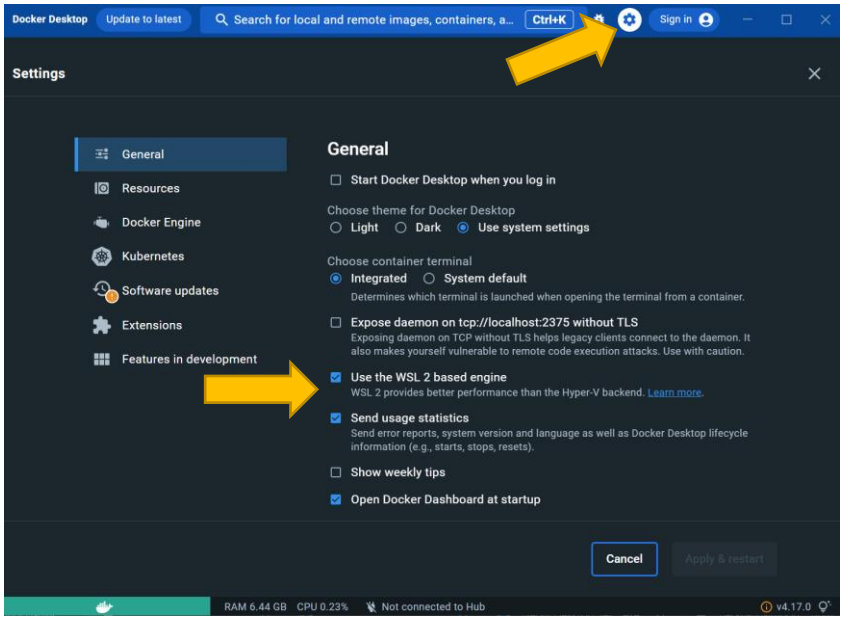
- https://gephi.org/users/download/

On Toledo / elsewhere, you will also have received download links to:

- twitch.cypherl – a dump of the database we will import into Memgraph
- twitch.graphml – a conversion of the full database dump to a GraphML file (which you can use in Gephi or NetworkX (Python))

Start Docker Desktop once it is installed. If running, you will see an icon in your task bar:

Windows users: you're best of the make sure the WSL2 backend is selected as the "engine" under settings:



You should also be able to run the `docker` command on a command line / Terminal window:

```
docker

Usage:  docker [OPTIONS] COMMAND

A self-sufficient runtime for containers

Commands:

[A long list of commands]

Run 'docker COMMAND --help' for more information on a command.
```

Keep your command line window open, and run the following command to start Memgraph (this is one single line):

```
docker run -it -p 7687:7687 -p 7444:7444 -p 3000:3000 -v mg_lib:/var/lib/memgraph -e MEMGRAPH="--query-execution-timeout-sec=3600" memgraph/memgraph-platform
```

What this means:

- docker run: run a command in a container
- -it: interactive mode: spawn a shell session
- -p: ports to forward from the container to your local machine
- -v: attach a volume
- -e: set an environment variable. We use this to set the query timeout to 1h (you can change this if you want)
- memgraph/memgraph-platform: the container to run, fetched from Docker Hub: https://hub.docker.com/r/memgraph/memgraph-platform

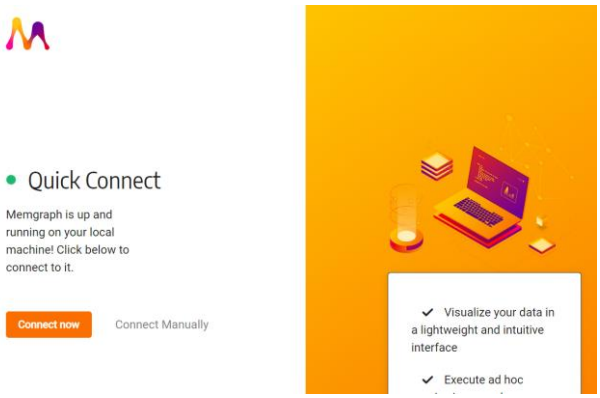
This will take a while the first time. You should see the following output:

```
Memgraph Lab is running at localhost:3000

mgconsole 1.3

Connected to 'memgraph://127.0.0.1:7687'

Type :help for shell usage

Quit the shell by typing Ctrl-D(eof) or :quit

memgraph>
```

Keep this window open to keep Memgraph running. You can enter Cypher commands at this console (make sure to end them with a semicolon ";"), but instead, navigate to:

- http://localhost:3000/

The following page should appear. Press "Connect now":



You won’t have any nodes or edges (yet):

Go to the “Import & Export” tab, and import the “twitch.cypherl” file. This will take a couple of minutes. You’re now ready to start exploring the data.

Working with the Graph

You can execute queries in the “Query Execution” pane.

Feel free to play around with some Cypher queries to explore this graph, e.g.:

```
match (s:streamer)-[e]-(t:tag) return * limit 100
```

Tip: it’s a good idea to use limit in order not to overflow yourself with results.

Tip: note that we labeled the edge (“e”) here, otherwise, they will not be shown in the result (this is different from the default behavior from Neo4j which connects nodes).

Tip: you can click nodes in the result view to see more info and expand them.

```
match (s:streamer)-[e]-(t:tag {name: "adhd"}) return * limit 100
```

Or:

```
match (s:streamer {name: "lirik"})-[e:recommends]-(t:streamer) return *
```

Or:

```
match (t:tag)-[e1]-(s:streamer)-[e2]-(g:game)
where s.description =~ '.*crazy.*'
return *
```

Or:

```
match (a:tag)<-[e*1..2]-(s:streamer)
where a.name = "ASMR"
return *
```

You can change size and color of the node types by going to the “Graph Style Editor” tab. E.g. after running this query:

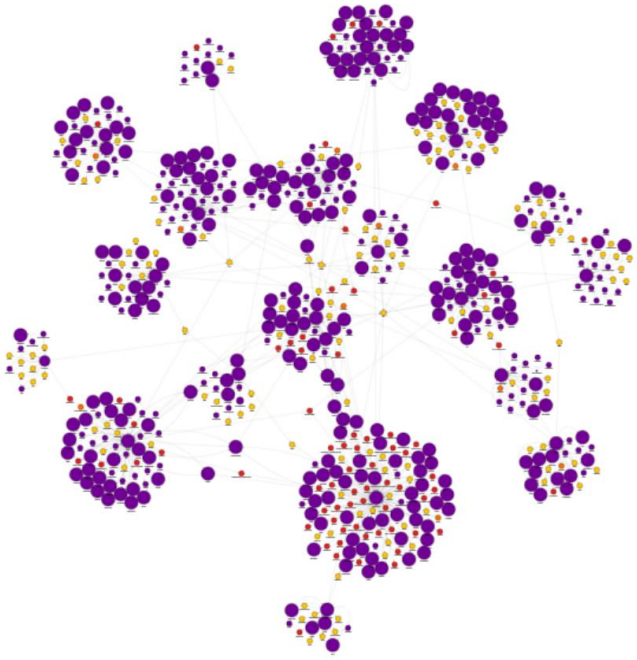
```
match (t:tag)<-[te]-(s:streamer)
where toLower(t.name) = "dad"
match (s)-[toe]->(to:tag)
match (s)-[ge]-(g:game)
match (s)-[re]-(s2)
return *
```

Tip: note that there’s a difference between using multiple match statements, using with, and comma-separating subgraphs in the same match line, e.g. see https://stackoverflow.com/questions/32742751/what-is-the-difference-between-multiple-match-clauses-and-a-comma-in-a-cypher-qu.

… you might notice that some nodes get the same color even although they are of different types, press the “Apply Default Style” button to reset all colors.

You can then play around by adding in more styles, e.g.:

```
Define(VIEWS_FIELD, "views_avg")
@NodeStyle And(
  HasProperty(node, VIEWS_FIELD),
  IsNumber(Property(node, VIEWS_FIELD))) {
  size: Min(AsArray(Mul(Property(node, VIEWS_FIELD), 2), 15))
}
```

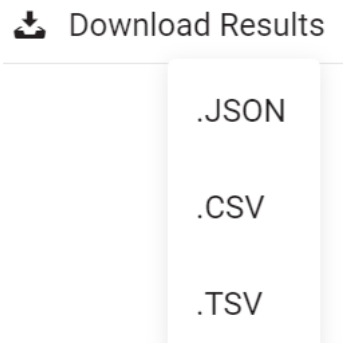


Memgraph's styling language is quite powerful: https://memgraph.com/blog/how-to-style-your-graphs-in-memgraph-lab.

Using Gephi

To load the data in Gephi, we must get it out of Memgraph in some manner. There is a Neo4j Gephi plugin, but it has not received much attention lately and is incompatible with Memgraph (even although every other Neo4j tool is).

In Neo4j, there is a custom APOC procedure to export queries as GraphML files, which Gephi can read, but sadly this is not present yet in Memgraph. Luckily, there is an export function to JSON:



This file can then be converted using the `json2graphml.py` Python script you will have received as well (note that you will need the `networkx` package):

```
python json2graphml.py "c:\users\seppe\downloads\memgraph-query-results-export.json"

Done, output file saved to: c:\users\seppe\downloads\memgraph-query-results-export.graphml
```

Finally, if you really want to load in the complete graph in Gephi, you can use the provided “twitch.graphml” file, though it will most likely be very cumbersome to work with, so perform initial filtering using Cypher first.

FYI: Some additional notes on exporting

The web interface provides an option to export a JSON file, but what about using Cypher directly. Currently, Memgraph provides the following options (https://memgraph.com/docs/mage/query-modules/python/export-util):

```
CALL export_util.json(path);
```

Which exports the full database, and:

```
CALL export_util.csv_query(path);
```

Which works e.g. as follows:

```
WITH "MATCH (s:streamer {name: 'xqc'})--(g:game) RETURN *" AS query
CALL export_util.csv_query(query, "/usr/lib/memgraph/query_modules/my_export.csv", True)
YIELD file_path
RETURN file_path;
```

It is recommended to run this in the terminal window rather than from the web interface.

```
memgraph> WITH "MATCH (s:streamer {name: 'xqc'})--(g:game) RETURN *" AS query
       -> CALL export_util.csv_query(query, "/usr/lib/memgraph/query_modules/my_export.csv", True)
       -> YIELD file_path
       -> RETURN file_path;

+-----------------------------------------------+
| file_path                                     |
+-----------------------------------------------+
| "/usr/lib/memgraph/query_modules/my_export.csv" |
+-----------------------------------------------+

1 row in set (round trip in 0.118 sec)
```

Memgraph recommends you use “/usr/lib/memgraph/query_modules/” as the path to export to. You might wonder how this can work on a Windows machine, and where your file actually is. The answer is it is inside of the docker container. So how to get it out? In another terminal, first run `docker ps` to figure out the container id (yours will be different):

```
docker ps

CONTAINER ID   IMAGE                     COMMAND                  CREATED       STATUS

f5d2ca848138   memgraph/memgraph-platform "/bin/sh -c '/usr/bi…"   5 hours ago   Up 5 hours
```

Next, we can copy the file as follows:

```
docker cp f5d2ca848138:/usr/lib/memgraph/query_modules/my_export.csv my_export.csv

Successfully copied 24.6kB to G:\my_export.csv
```

The file is then present on your local environment, though is not easily parsed – so it’s probably better to export using the web interface or use the provided “graphml” file (which you can easily load in with `networkx` should you want to).