Social Network Analysis of Twitch Gamers

Using GraphX and PySpark

CS 631 Final Project Report

Hongyu Chen
20792778
h542chen@uwaterloo.ca

Muhammad Furqan
20858360
mfurqan@uwaterloo.ca

Marcus Huang
21061943
m43huang@uwaterlo.ca

ABSTRACT

Social Networks have become a dynamic ecosystem, where connections, interaction, communication and influence thrive in the digital world. Among all of these networks, Twitch is one of the most famous gaming communities. It has fostered vibrant communities where gamers are able to make connections and collaborate with each other.

Social Network Analysis has immense significance in today's world. It will provide valuable insights into relationships between individuals in the network, and allow researchers and developers to understand the network, identify the key players and optimize the performance of the network.

Our project aims to understand these connections, which is essential for comprehending gamer behaviour and also identifying influential gamers in the community. We will dive into the social network analysis within the Twitch gamer community. Using GraphFrame, a powerful processing framework to unravel the structures of the Twitch community. Through social network analysis, we aim to identify the influential gamers and quantify user importance. Which in the end shows the relationships within the gaming community.

Dataset

https://snap.stanford.edu/data/index.html#socnets

Twitch Gamers Social Network (168K nodes, 6.8M edges): a social network of Twitch users which was collected from the public API in Spring 2018.
https://snap.stanford.edu/data/twitch_gamers.html

KEYWORDS

Spark, GraphFrames, PageRank, BFS, Social Network Analysis,

1 Project Goals and Learning Objectives

Our project will focus on implementing graph algorithms using Graph Frame to analyze the network of Twitch gamers. The main learning objective includes computing PageRank for gamers given the connections between each other, measuring the importance of each gamer, and identifying the most connected (influential) gamers in the graph (community). Compute the degree of separation for a single gamer or between two defined gamers using Breadth First Search (BFS).

We will implement both convergence and iterative versions for PageRank to compute the PageRank scores for each gamer. With a given gamer, BFS is implemented to calculate the number of steps required to reach all other gamers. Similarly, the number of steps needed to reach from one gamer to another is also computed using BFS.

Overall, the project goal is to analyze the Twitch gamers community using GraphX. The main object is identifying influential gamers and employing pathfinding algorithms within the network.

2 Methodology

Spark 3.4.2 [1]
Apache Spark is a unified analytics engine for large-scale data processing. It provides high-level APIs in Java, Scala, Python and R, and an optimized engine that supports general execution graphs.

Graphframes 0.8.3 [2]
This is a prototype package for DataFrame-based graphs in Spark. Users can write highly expressive queries by leveraging the DataFrame API, combined with a new API for motif finding. The user also benefits from DataFrame performance optimizations within the Spark SQL engine.

2.1 Methodology for PageRank

PageRank[4] is an algorithm famously employed by Google Search to determine the ranking of web pages within their search engine results. In our case, we adapt PageRank to assess the prominence of Twitch gamers.

In this framework, the voting weight of each link (representing connections between gamers) is proportionate to the significance (PageRank) of the originating gamer. Suppose gamer j, has a significance voting (PageRank) of r, and it possesses n out-links (out-connections), each link will receive r/n votes.
The overall importance of gamer j is derived from the cumulative votes it receives from its in-links (in-connections).

During each iteration, the random surfer faces two choices:
With a probability of β, the surfer follows a link at random.
With a probability of 1-β, the surfer jumps to a random page.

PageRank equation is :

$$r_j = \sum_{i=1}^{j} (1 - \beta)\frac{r_i}{d_i} + \beta\frac{1}{N}$$

N stands for the number of vertices (gamers)..
To solve dead ends, we pre-process to replace dead ends with self-loops.

Key functions:

page_rank(graph, resetProbability=0.15, sourceId=None, maxIter=None, tol=None)

graph is a GraphFrame object with edges as connections between gamers and vertices are distinct individual gamers. Edges and Vertices are data frame objects read directly from the dataset.

resetProbability is the probability of jumping to some random gamers (PageRank scores).

sourceId in the graph is designated as the source node. It indicated that we are doing a personalized page rank instead of an ordinary page rank. Personalized page rank **[5]** is performed with respect to that source node. It is initialized by assigning all probability mass (1.0) to the source node, and none to the other nodes. In contrast, ordinary page rank is initialized by giving all nodes the same probability mass. Whenever a personalized page rank makes a random jump, it jumps back to the source node. In contrast, ordinary page rank may jump to any node. In personalized page rank, all probability mass lost dangling nodes are put back into the source nodes. In ordinary page rank, lost mass is distributed evenly over all nodes.

maxIter=None, tol=None Exactly one of them should be set to a non-None value. If maxIter, the PageRank will run for a fixed number of iterations. If tol, we will iterate page rank until the maximum change is less than a specified threshold (tol), i.e., until all nodes' page ranks have converged.
PageRank function is implemented using **graphframes.lib.Pregel [3]** as the core function. Which is a Pregel-like bulk-synchronous message-passing API based on DataFrame operations.
The key operations are the following:
setMaxIter(value): Sets the maximum number of iterations. In the iterative version of the PageRank, it is used to determine to halt the processing if the max interactions are reached. In the convergence version, we will set it as 1. Since we need to check for convergence after each iteration.
withVertexColumn(colName, initialExpr, updateAfterAggMsgsExpr): Defines an additional vertex column at the start of the run and how to update it in each iteration. In PageRank, the **initialExpr** is 1 / number of gamers for the ordinary PageRank and 1 for the personalized PageRank. **updateAfterAggMsgsExpr** is the expression after we receive messages from other vertices (neighbour).
To explain the expression, we will introduce:
sendMsgToDst(msgExpr): This defines a message to send to the destination vertex of each edge triplet. It sends (the current vertice i 's PageRank) / (its outdegree to the neighbour). In the equation above, it represents (ri / di)

Then we can go back to **updateAfterAggMsgsExpr**, after receiving a message from the neighbour "(ri / di)", the update expression is "(ri / di)" * (1-β) + β / N if we are doing ordinary PageRank, the update expression is "(ri / di)" * (1-β) + β if i is the source vertice, "(ri / di)" * (1-β) if i is not the source vertice if we are doing personalized PageRank.
aggMsgs(aggExpr): Defines how messages are aggregated after being grouped by target vertex IDs. In our PageRank, we do a summation after grouping which completes the equation.

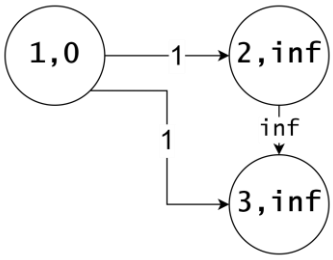
In summary, the page_rank function computes the PageRank scores for each gamer using graphframes.lib.Pregel. In this process, every vertex forwards its voting weight to its neighbouring vertices, with the weight being contingent on the vertex's outdegree. Subsequently, upon receiving all votes from its neighbours, each vertex calculates its individual PageRank.

2.2 Methodology for BFS

BFS is a fundamental graph traversal algorithm, which explores a graph in layers, starting from a given source vertex and moving outward. Because of this, BFS ensures that the first time a vertex is visited, it is reached via the shortest path, in terms of number of edges, from the source. BFS has been widely incorporated in many applications involving graphs like Single-Source Shortest Paths and Connected Components. In this project, we utilize BFS based algorithms to implement the following operations on the social graph we proposed:

1. Degree of separation for a single user: Determine the number of steps required to reach all other users from a given user using Breadth-first search (BFS).
2. Degree of separation between two defined users: Calculate the number of steps required to traverse from one user to another using BFS
3. Single-Source Shortest Path problem using Dijkstra's algorithm, which finds the path between two vertices in a graph with the minimum number of edges.

To implement the base BFS algorithm for our project, we used the Pregel-like API provided in the graphframes, **graphframes.lib.Pregel [3].**
A Pregel run implemented by the graphframes mainly involves initialization and iterations: when a run starts, it expands the vertices in the DataFrame using column expressions defined by withVertexColumn(), in our base BFS implementation, we initialize the distances to the source edge with the value of infinity, except the source node itself is initialized to 0.



At each Pregel iteration, there are 3 phases:

1. Given each edge triplet, it generates messages and sends them to specified targets. In our implementation, the BFS behaviour is described by **sendMsgToDst()**: we check the distance to the source (root) node of the current candidate path (from src node to the dst node of the current message) against the pre-existing distance recorded in the dst node, a message containing the updated distance is sent out if the current candidate path is shorter, otherwise the message will retain the pre-existing value of the minimum distance.
2. Aggregate messages by target vertex IDs, described by aggMsgs(): in our use case, we aggregate messages by selecting the messages containing the minimum distance to the source for each node.
3. Update additional vertex properties based on aggregated messages and states from previous iteration, described by withVertexColumn(): the update for the current iteration coalesces the message from previous iteration with the value of infinity to account for the nodes that has not been reached in the previous iteration, so that such nodes will have a distance value of infinity.

Similarly, a modified version of BFS is also implemented to facilitate the operation that recovers the shortest path from the source node. To facilitate the path recovery, we made the following modifications:

1. The new column **path** is initialized to **null** with the exception of the source node being initialized to an empty array.
2. A message containing the candidate path is generated if the current candidate path is shorter than the path currently in the dst node, or the current dst node has no recorded path.
3. The message is now aggregated by selecting the last generated message, since the path length is defined as the number of edges thus all the new messages generated will have the same length (shorter ones would not be generated due to the check in the message generation process, since it would have been already recorded in previous iterations if it were an updating path).
4. Updating the vertex column is modified to accept the new path if there is any generated, otherwise retain the current path.

Implemented Operations:

degree_of_seperation(graph, id_from, id_to, max_iter=10)
The operation obtains the degree of separation between 2 gamers.
graph is a GraphFrame object with edges as connections between gamers and vertices are distinct individual gamers. Edges and Vertices are data frame objects read directly from the dataset.
id_from is the id of the source gamer that we are interested in.
id_to is the id of the target gamer that we are interested in.
max_iter is the maximum number of iterations, defaults to 10.

degree_of_seperation_single(graph, id, max_iter=10)
The operation obtains the degree of separation from a single source gamer to any other gamer.
graph: same as above.
id is the id of the source gamer that we are interested in.
max_iter: same as above.

sssp(graph, id, max_iter=10)
The operation obtains a shortest path from a single source gamer to any other gamer.
graph: same as above.
id is the id of the source gamer that we are interested in.
max_iter: same as above.

3 Outcome

3.1 Outcome for PageRank

Insights about Twitch Gamers community from Page Rank:
Most of the gamers in the community have very low PageRank values, about 80% of them. Only about 20% of gamers have a bit higher page rank and less than 1% of gamers have incredibly high Page Rank values (more followers), which we can call top gamers. This result is expected and not surprising that only a few gamers stand out from the crowd much more.
The result below is using a max iteration set to 5 and no source id or tolerance value set.

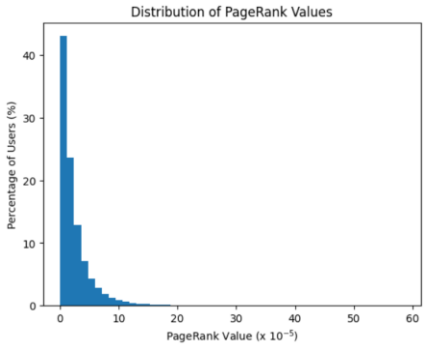


Figure: Distribution of PageRank values across entire dataset

Overcoming Challenges in Implementing Page Rank:
The PageRank algorithm was challenging to implement due to its iterative nature and the large size of the Twitch network. We faced challenges in ensuring the convergence of the algorithm, which we addressed by carefully choosing the tolerance value.

One of the main issues was handling dead ends in the graph. Dead ends are nodes that have no outgoing edges, which can cause the loss of PageRank scores. We handled this by replacing dead ends with self-loops. This method involves adding a self-loop to each dead end, effectively turning it into a node with one outgoing edge.

Another challenge was understanding the implementation of the built-in PageRank algorithm in GraphFrames. We noticed discrepancies between the output of our implementation and the output of GraphFrames' PageRank. This led to confusion about whether GraphFrames initializes each node with 1 instead of 1/num_node and whether Graph.pregel sends values synchronously to every node.

Despite these challenges, we were able to implement a version of the PageRank algorithm that allowed us to proceed with our analysis of the Twitch gamers.

3.2 Outcome for BFS

Evaluations on execution efficiency:
All evaluations are conducted on a single machine, with a maximum number of iterations set to 5.

Degree of Separation

Dataset	Execution Time (s)	Peak Memory (MiB)	Execution Time (s) - Naive	Peak Memory (MiB) – Naive
Reduced	62.82	67.9	0.01	0.17
Full	166.62	258.5	75.77	542.52

Single-Source Shortest Path

Dataset	Execution Time (s)	Peak Memory (MiB)	Execution Time (s) - Naive	Peak Memory (MiB) - Naive
Reduced	72.71	99.0	0.01	311.5
Full	164.56	366.3	64.93	566.41

On the reduced dataset, the parallelized approach takes longer but consumes less memory. The naive approach achieves near-instantaneous execution but exhibits a substantial increase in memory usage.

On the full dataset, the parallelized approach for the full dataset is time-intensive but maintains moderate memory usage. The naive approach reduces execution time but demands significantly higher memory.

From above results, the execution time for the naive (non-parallelized) method seemed to have an advantage in terms of execution time. But we can see from the comparison between the reduced and the full dataset, the performance (in terms of execution time) degraded significantly worse compared to the parallelized method (from almost instantaneous to about a minute versus from over a minute to over 2 minutes). Which signifies the scalability for the naive method to be extremely limited that it might render the whole method intractable for data of larger scale. Also notice that the naive method generally requires more memory than the parallelized method.

Due to the limited availability of computational resources, the evaluations were all conducted on a single machine which further limits the demonstration of the full potential of the parallelized method.

Insights about Twitch Gamers community from BFS:
The maximum degree of separation between any two connected gamers is 5, with the average being 2.80 and most of the gamers are separated by a degree of 3.
Most of the gamers in the community are connected to at least one other gamer, specifically, 167894 out of 168114 (99.87%).

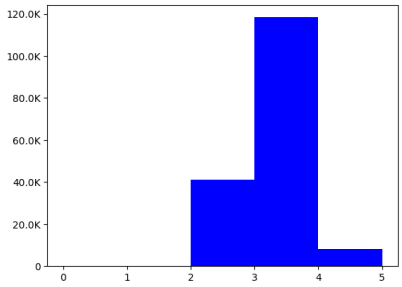


Figure: Histogram of the Degree of Separation

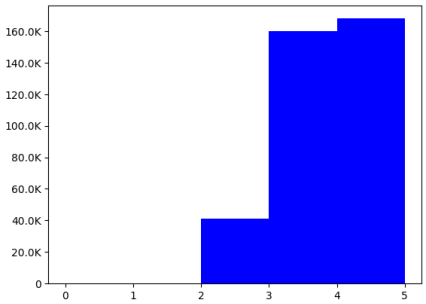


Figure: Cumulative Histogram of the Degree of Separation

Overcoming Challenges in Implementing BFS:

During the BFS algorithm implementation, our team encountered a significant hurdle related to documentation. Specifically, the absence of GraphX, which is not supported within the PySpark, posed difficulties. Additionally, the alternative library we opted for, graphframes, lacked proper documentation. Especially for the Pregel functionalities we heavily relied on for implementing multiple operations in this project. This lack of clear guidance impacted our development process and required us to rely heavily on experimentation and community forums to overcome the challenges.

3.3 Outcome for Language Network

Other than Breadth-First Search (BFS) and PageRank, a language network was also constructed using GraphFrames and PySpark. This network was built from a dataset of Twitch gamers, with the aim of identifying communities of gamers who speak the same language.
The vertices of the graph are created by selecting distinct gamers from the edges DataFrame. A new DataFrame is created where each edge represents a connection between users who speak the same language. A GraphFrame is then constructed using the vertices and edges, and the connected components of this graph are found. The code then calculates the total number of communities and users for each language.

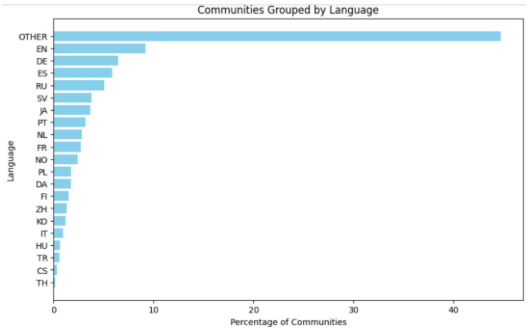


Figure: Percentage of communities grouped by Language

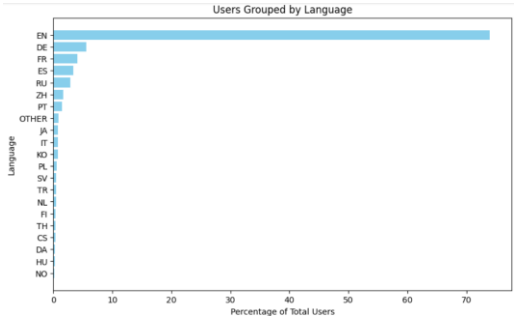


Figure: Percentage of gamer users by Language

The charts above show that over 70% of gamers are in English (EN) communities followed by German (DE), French (FR), Spanish (ES) and Russian (RS) at about 4% to 5% each. This result is expected since English dominates globally as the most used international language.

4 Comparison

Comparison of the PageRank and Breadth-First Search (BFS) algorithms:

Both algorithms provide measures of importance within the network, but they do so in fundamentally different ways. PageRank is a global measure that takes into account the entire structure of the network. It assigns a numerical weight to each user in the network, with the purpose of measuring its relative importance within the network. A user can have a high PageRank either by having many links or by being linked by other users with high PageRank.

On the other hand, BFS is a local measure that is based on the proximity of users to a particular user. BFS explores the network from a given source user and visits all its neighbors before visiting the neighbors' neighbors. BFS can be used to measure the closeness or distance between two users in the network.

Differences in the gamers identified as influential by the PageRank and BFS algorithms:

Degree of separation between top 10 PageRank gamers and bottom 10 PageRank gamers using BFS and PageRank algorithms together:

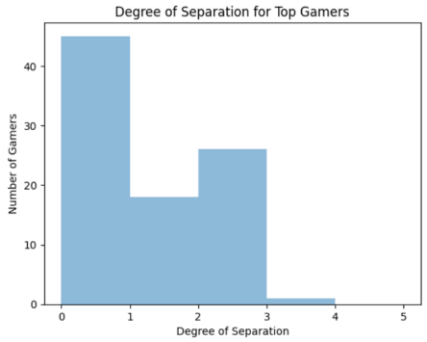


Figure: Degree of separation between bottom 10 PageRank

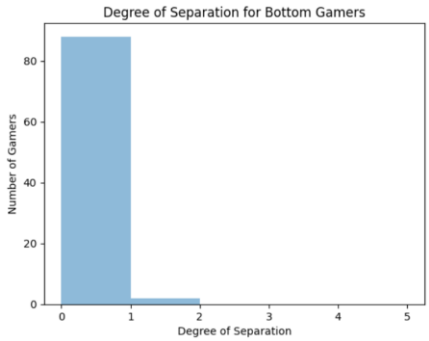


Figure: Degree of separation between top 10 PageRank

The histogram clearly illustrates the difference in connectivity between the top and bottom gamers based on PageRank. The first bar, representing a degree of separation of 0, indicates that the majority of the bottom gamers are not connected to each other ('infinite separation').

In contrast, the top gamers show a higher degree of connectivity between each other, with about half of them having a degree of separation between 1 and 3. This suggests that top gamers tend to form more interconnected communities, which could be another factor contributing to their high PageRank scores as well.

5 Conclusion

This project has been a useful journey in understanding and applying social network analysis and graph processing techniques. The main learning objectives were to compute PageRank for gamers given the connections between each other, measure the importance of each gamer, identify the most connected gamers in the community, compute the degree of separation for a single gamer or between two defined gamers using Breadth First Search (BFS).

The project successfully achieved these objectives, providing valuable insights into the Twitch gamers community. The PageRank algorithm identified influential gamers based on their connections within the network, while BFS provided a measure of proximity between gamers. The analysis revealed that most gamers in the community have very low PageRank values, indicating that only a few gamers stand out significantly. The BFS results showed that most of the gamers are separated by a degree of 3. The Language Network showed that the majority of gamers are in English (EN) communities. Furthermore, we found that top gamers tend to form more interconnected communities compared to bottom gamers.

For future work, there are several potential directions to explore. These include analyzing user engagement, maturity level of content, account lifespan, prevalence of inactive accounts, and the influence of affiliates within the network. Additionally, more granular communities within specific language groups could be identified, and correlation analysis between these features and the measures of importance obtained from the PageRank and BFS algorithms could be performed. These analyses could provide a richer understanding of the Twitch gamers community and reveal new insights into user behavior, content preferences, and network dynamics.

In conclusion, this project has demonstrated the power and potential of social network analysis and graph processing techniques for understanding online gaming communities. The methodologies and results of this project can provide valuable insights for researchers, developers, and the gaming community.

REFERENCES

[1] Spark Release 3.4.2 https://spark.apache.org/releases/spark-release-3-4-2.html
[2] Graphframes 0.8.3 https://github.com/graphframes/graphframes/releases
[3]pygraphframes 0.8.3 documentation https://graphframes.github.io/graphframes/docs/_site/api/python/index.html
[4] PageRank https://student.cs.uwaterloo.ca/~cs451/slides/05%20-%20Graphs.pdf
[5] Personalized page rank https://student.cs.uwaterloo.ca/~cs451/assignment3-431.html
Github repository link for this project: https://github.com/chenho2000/CS631-Final-Project