CSC494 Final Report – Community Expansion Algorithm for Twitter Social Network

Yanke Mao

April 25, 2023

1. Introduction

In this project, we aimed to develop a novel community expansion algorithm for twitter social network based. It would generate an expanded user list for a certain community after given an initial core user list that was the core users of this community introduced by Peter Marbach in the paper “Structure of Core-Periphery Communities”.

1.1 Background

The full algorithm should consist of two parts: the core detection part and the community expansion part. The core detection part aimed to generate a reasonable core user list after given a starting user in the community. The community expansion part would work on the output from the core detection and generate the expanded community.

The community expansion part consists of the core refiner algorithm and the expanding community algorithm. The core refiner aimed to refine the core user list based on the initial core users given by core detection. The community expansion aimed to expand the community based on the refined core user list. We will introduce them in detail later in the report.

Next, we want to introduce some concepts we are going to use in the following report:

- **Friends:** The followings of a twitter user
- **Followers:** The followers of a twitter user
- **Respecting user list:** The user list that we score a user respecting to.
- **Initial core users:** The core users from the core detection output.
- **Refined core users:** The refined core users from the core refiner algorithm.
- **Influence One Score, Influence Two Score, Production Score, Consumption Score:** Four scores to evaluate the influence and importance of a user in the community. We will define them later.
- **Intersection Ranking:** The ranking method that we used to rank users based on four scores.

1.2 Existing Ranking Method

There were four scores to evaluate the influence and the significance of a user in the community, which were influence one score, influence two score, production score, and consumption score. They were defined as below:

- **Influence One Score:** The number of tweets from a user U retweeted by the user U's direct followers in the respecting user list + The number of tweets retweeted by U and later retweeted by U's direct followers in the respecting user list. The score will be divided by number of tweets and retweets posted by U.
- **Influence Two Score:** The percentage of retweets roughly credited to U in all retweets from U's direct followers. The retweets roughly credited to U mean the tweets that were counted in Influence One Score. The score is the sum of the percentages of all U's direct followers in the respecting user list.
- **Production Score:** The number of retweets made by u's followers in the respecting user list.
- **Consumption Score:** The number of retweets u made from u's friends in the respecting user list.

Based on these four scores, we were able to rank users in the community by their significance. We will use intersection ranking in our project.

- **Intersection Ranking:** Ranking users in the community based on their worst ranking among the four rankings respecting to four scores respectively. For example, if user U has 4^{th} influence one score, 5^{th} influence two score, 6^{th} production score, and 7^{th} consumption score, the algorithm will consider this user has a ranking score of 7, and the intersection ranker would rank him/her based on this score which was 7.

From previous work, these scoring functions and ranking functions were proven effective in most cases. But there are still some problems of the algorithm waiting for us to fix.

1.3 Existing Problems

There were several problems for the community expansion part showed by previous work:

- The existing algorithm spent extremely long running time, which would cost about 12 hours to run the algorithm once.
- There were some significant errors in the existing algorithms implementation that made the algorithms not do their jobs.
- After expanding the algorithm, the ranking for some of the initial core users and the refined core users dropped sharply in the expanded community. However, the core users were expected to keep high ranking along the whole process as they were the core of the community.

1.4 The Improvement in this Semester

- Updated the code environment setup guideline.
- Reduced the running time of the algorithm from about 12 hours to about 10 minutes.
- Fixed all errors in the existing algorithm.
- Adapting the scoring function to make them more reliable.
- Improve the structure of the threshold of the community expansion part.
- Introduce the respecting user list for the algorithm.
- Adding useful tools for data analysis.

2. Environment Setup (draft)

The code was provided on the Github: https://github.com/SNACES/core.
PLEASE follow the guideline below:

- **Download the MongoDB Compass.** We will use the MongoDB to manage the user data we downloaded from Twitter API. This would also enable you to view and manage the data directly from the database.
- **Clone the code from the Master branch on Github.** It is suggested to create a new virtual environment with Python 3.8 for the project, since there may be some conflicts between packages that you installed.
- **Apply a Twitter Developer Account on https://developer.twitter.com/en/portal/dashboard.** Since we need to download the user information from Twitter API, you will need a developer account. Since twitter has changed its API service charging policy, the free API service may not enough for the project. It is recommended to apply for a academic research access for Twitter API. After you got an account, you can get four values API Key, API Key Secret, Access Token, and Access Token Secret that we need in the next step.
- **Create a folder with path ./log.** We will store the log file in this folder.
- **Create a file with path ./conf/credentials.py.** You should fill the file in the following format:

```
ACCESS_TOKEN = "<Your Access Token>"
ACCESS_TOKEN_SECRET = "<Your Access Token Secret>"
CONSUMER_KEY = "<Your API Key>"
CONSUMER_SECRET = "<Your API Key Secret>"
```

- **Run the install script ./install.sh.** If it doesn't work, you can run each command in the terminal manually instead.
- **Run python ./setup.py to setup the Pipfile.**
- **Run pipenv shell to start a pip environment using the pip file.**
- **Be sure to install the correct version for each package.** It is because that for some packages, the newest version wasn't compatible with our program. You can check the correct version in the ./requirements.txt.

To run the program, first run pipenv shell to start a pip environment. Then run python ./SNACES.py. You should choose 6 to run the community expansion algorithm. After the program is done, you can find the result for the final

expanded community as well as the result for each iteration in the core refiner and the community expansion.

3. Code Structure:

3.1 Scoring function

As defined in the introduction part, the old scoring function consists of four scores, which were influence one score, influence two score, production score, and consumption score. You can see more details about the old scoring function in the report for the core detection part.
To make the scoring function more reliable, we changed the scoring function a little bit as the below pseudo code.

Algorithm 1 New Scoring Function

1: *Scoring user* ← the user we want to score
2: *Respecting user list* ← the user list that we will score respecting to
3: *New score* ←
Old Score / len(respecting user list except the scoring user)
4: **return** *New Score*

This new algorithm solves the unfair problem between the cases that the scoring user belongs the respecting user list and the scoring user doesn't belong to the respecting user list. For example, if there were 10 users in the respecting user list, and the scoring user A is one of them, while scoring user B is not one of them. When we score the production score which is the sum of retweets from A/B's followers in the respecting list, the user A can only have maximum 9 possible followers in the respecting user list, while the user B can have maximum 10 possible followers in the respecting list, which means the user B has advantages compared to A. By reweighting the score based on the number of the real respecting user list (the respecting list except the scoring user), this unfair would be solve.

3.2 Tool function for community expansion

The pseudo code for two major roll function for community expansion part was shown below.

Algorithm 2 Select Potential Candidates

1: *Community* ← Group of users in the current community
2: *T* ← *len*(*Community*) ∗ *Threshold_1*
(the minimum threshold for the number of followers in the current Community)
3: *P* ← *Maximum* Number of Candidate we want (Default 200)
4: *IsMore* ← *True* (Whether there are more potential candidates)
5: *Map* ← Initialize an empty map of user with score
6: *Candidates* ← Initialize an empty list of users

```
7: for each U in Community do
8:     Friends ← users that U follows
9:     for each F in Friends do
10:        if F is not in Community then
11:            Increment Map[F ] by 1
12:        end if
13:    end for
14: end for
15: RevMap ←Switch Map such that it becomes a map of score with users
16: for each Score in RevMap do   (Sorted in descending order)
17:     if Score ≥ T and len(Candidates) + len(RevM ap[Score]) ≤ P then
18:         Add RevMap[Score] to Candidates
19:     else
20:         if Score < T then
21:             IsMore = False
                    (Traversed through all candidates above Threshold)
22:         end if
            break
23:     end if
24: end for
25: return Candidates, IsMore
```

The select potential candidates function would select the potential candidates for the expanded community based on the friends of users in the current community. It was given the current community, the bottom threshold of the number of followers of the candidate in the current community, and the maximum number of potential candidates we may considered in each calling. The function would search for the friends of the current community users and sorting them by the number of their followers in the current community (also called local followers). If the number of their local followers reach the bottom threshold, they would be added into the potential candidates list until the number of potential candidates reach the maximum number of candidates that we want. Finally, the function would return the list of the potential candidates as well as whether we have considered all qualified potential candidates in the current community.

Algorithm 3 Filter Candidates

```
1: Community ← Group of users in the current community
2: Candidates ← Group of potential candidates
   Respecting User List ← The user list that we score users respecting to
3: T_Influence1 ← Average Influence 1 of Top X users in Community
4: T_Influence2 ← Average Influence 2 of Top X users in Community
5: T _Prod ← Average Production of Top X users in Community
6: T _Con ← Average Consumption of Top X users in Community
```

7: *T _Follower_ceil* ← Average Number of global Followers of Top X users in *Community*
T _Follower_floor ← Average Number of global Followers of Bottom X users in *Community*
8: *NewCandidates* ← Initialize an empty list for filtered candidates
Num_new_members ← The maximum number of new member we want
9: **for** each *U* in *Candidates* **do**
10: *U _Influence*1 ← Influence 1 of *U* respecting to Respecting User list
11: *U _Influence*2 ← Influence 2 of *U* respecting to Respecting User list
12: *U _Prod* ← Production of *U* respecting to Respecting User list
13: *U _Con* ← Consumption of *U* respecting to Respecting User list
14: *U _Follower* ← Global Followers *U*
15: **if** *U_Influence*1 ≥ *T_Influence*1 * *Threshold_1*
and *U _Influence*2 ≥ *T_Influence*2 * *Threshold_1*
and *U_Prod* ≥ *T_Prod* * *Threshold_1*
and *U_Con* ≥ *T_Con* * *Threshold_1*
and *U_Follower* < *T_Follower_ceil* * *Threshold_2*
and *U_Follower* ≥ *T_Follower_floor* * *Threshold_3*
then
16: **Add** *U* to *NewCandidates*
17: **end if**
18: **end for**
19: *C_Influence*1 ← Influence 1 Ranking respecting to Respecting User list
20: *C_Influence*2 ← Influence 2 Ranking respecting to Respecting User list
21: *C_Prod* ← Production Ranking respecting to Respecting User list
22: *C_Con* ← Consumption Ranking respecting to Respecting User list
23: *Ranking* ← Initialize an empty map for new candidate ranking
24: *RevMap* ← Switch *Map* such that it becomes a map of score with users
25: **for** *i* in *range*(*len*(*C_Prod*)) **do** *(Start from Rank 1 to the last rank)*
26: *Intersection* ← users that are higher than Rank *I* in all of *C_Influence*1, *C_Influence*2, *C_Prod*, *C_Con*
27: **for** *U* in *Intersection* **do**
28: **if** *U* not in *Ranking* and *U* not in *Community* **then**:
29: *Ranking*[*U*] = *i*
30: **end if**
31: **if** *len*(*Ranking*) > *Num_new_members* **then**:
32: **break**
33: **end if**
34: **end for**
35: **end for**
36: **return** List of users in *Ranking*

The filter candidates function would be given the threshold for filter candidates, the current community user list, the potential candidates list the number of maximum filtered candidates we want, and the respecting user list.

The function will find candidates whose scores and the number of global followers satisfy the threshold showed in the red code part, rank them by the intersection ranking method, and then return the top *Num_new_members* of the ranked qualified candidates as the filtered candidates. These filtered candidates would be added into the community.

There's another strategy to filter the candidates, which is for each user, we will choose his relatively better scores from influence one score and influence two score as the unique influence score. The unique influence score would replace the influence one and influence two. We will discuss this strategy in detail in the experiments part.

3.3 The main structure of the community expansion part

The whole community expansion part will first call core refiner given the initial core users from the core detection part to refine the core user list, and then call the community expansion algorithm to expand the community based on the refined core user list. we are going to introduce the main body of the core refiner and the community expansion algorithm.

Algorithm 4 Core Refiner

```
1: Community ← initial core users
   Respecting list ← initial core users
2: PrevCommunity ← []
3: CoreSize ← Max Number of Core Users we want (Default 15)
   Max_loop ← Max Number of loop
   Loop ← 0
4:   while PrevCommunity == Community and Loop < Max_loop:
5:        curr_candidate ←Select Potential Candidates(Community)
6:        final_candidates ←
              Filter Candidates(curr_candidate, Community, Respecting list)
7:        new_community ←Community + final_candidates
8:        PrevCommunity ← Community
9:        Community ← the top CoreSize Users in rank(new_community)
          Loop ← Loop + 1
10: return Community
```

The core refiner would be given the initial core users from the output of the core detection part, the maximum number of iterations, and the core size that we want for the refined core user list. The function would call find candidates function and filter candidates function iteratively, until the refined user list becomes stable or reach the maximum iteration times.

In the code above, the respecting user list was set to the initial core users, and it was consistent along the whole process. However, there's another strategy

to set the respecting user list. That is making the respecting user list always the same as the previous community, which means the respecting user list would update after each iteration. We will discuss about both of these two strategies in the experiments part.

Algorithm 5 Community Expansion

```
Community ← Refined core user list
  Respecting list ← Refined core user list
PrevLength ← Number of users in Community
P ← Number of Candidate we want        (Default 500)
IsMore ← True      (Whether there are more potential candidates)
while IsMore or PrevLength != len(Community) do
        if PrevLength == len(Community) then:
            Increment P by 200
            (If no one is added last time, but there are more user to check)
        end if
        PrevLength ← Number of users in Community
        C_Influence1 ← Influence 1 Ranking respecting to Respecting list
        C_Influence2 ← Influence 2 Ranking respecting to Respecting list
        C_Prod ← Production Ranking respecting to Respecting list
        C_Con ← Consumption Ranking respecting to Respecting list
        Community ← Community ranked by intersection ranking
        Candidates, IsMore ← Select Potential Candidates(Community, P)
        NewCandidates ←
                    Filter Candidates(Candidates, Community, Respecting list)
        Community = Community + NewCandidates
end while
return Community
```

The main logic of the community expansion algorithm is mainly the same as the logic of the core refiner algorithm, except the community expansion has a relaxed threshold. Besides, it was given the refined core user list as the respecting list, while the core refiner was given the initial core users as the respecting list.

Same as the core refiner, the respecting user list can also be replaced by the previous community in the community expansion algorithm. We will discuss this in detail in the experiments part.

4. Experiments:

In this report, we mainly focused on the chess community. We include four major experiments:

1. The experiment to confirm the reliability of initial core user list.

2. The experiment to choose the suitable respecting list for community expansion algorithm.
3. The experiment to choose the suitable respecting list for core refiner algorithm.
4. The experiment to decide whether combining the influence one score and influence two score together.

4.1 Confirm the reliability of the initial core user list.

From the core detection part, we were given an initial core user list with ten users as below:

rank	userid	username	influence c	influence t	productior	consumpti	local follov	local follov	global foll	global foll	is new user
0	1.33E+18	ChampChe	0.073361	0.593883	317	320	7	9	23405	430	1
1	2.23E+09	PlayMagnu	0.117582	0.219749	131	176	8	0	13451	635	1
2	3.16E+09	GrandChes	0.064237	0.232999	149	133	7	9	30989	268	1
3	2.57E+08	anishgiri	0.045528	0.09013	81	55	7	8	273709	287	1
4	1.88E+09	DavidHow	0.153502	0.086081	114	97	7	9	29442	369	1
5	23612012	STLChessC	0.033142	0.156591	126	173	7	9	37487	1708	1
6	20745074	chesscom	0.037343	0.114989	67	20	7	9	444267	524	1
7	1.33E+08	TarjeiJS	0.034149	0.07261	31	53	7	9	26430	2657	1
8	60494861	Bennyficial	0.017751	0.030337	6	27	7	0	8454	2029	1
9	2.5E+08	FIDE_chess	0.008826	0.040459	52	20	7	9	216093	1534	1

Since the users ‘anashgiri’, ‘chesscom’, and ‘FIDE_chess’ have comparatively large number of global followers compared to other initial core user, and ‘Bennyficial’ has a comparatively small number of global followers, we removed them from the core user list manually, since we expect the initial core users have the similar characteristics on the global followers. Then, the rest six initial core users were divided into three different types:

Platforms: ChampChessTour, PlayMagnus, GrandChessTour, STLChessClub

Grandmaster: DavidHowellGM

Journalist: TarjeiJS

We want to ensure whether these different types of initial core users all belong to the same chess community. Therefore, we run the core refiner algorithm with these three types of users respectively to compare whether the refined core user lists are the same or not. When the core size was set to 10 and the respecting user list was replaced by the previous community, all these three types of users would return the same refined core user list as below.

rank	userid	username	influence o	influence t	productior	consumpti	local follow	local follow	global follc	global follc	is new user
0	1.33E+18	ChampChe	0.093738	0.505926	405	263	7	9	23405	430	0
1	2.23E+09	PlayMagnu	0.130893	0.220385	145	182	8	0	13451	635	0
2	3.16E+09	GrandChes	0.065364	0.239263	148	125	6	9	30989	268	0
3	1.88E+09	DavidHow	0.174367	0.088151	150	97	7	9	29442	369	0
4	23612012	STLChessC	0.045654	0.239239	164	179	7	8	37487	1708	0
5	1.26E+08	USChess	0.036464	0.0782	64	56	7	9	40289	1450	0
6	1.87E+08	ginger_gm	0.051069	0.107831	36	181	7	0	19955	193	0
7	4.37E+09	chessable	0.027431	0.09744	114	123	7	8	30030	2022	0
8	1.33E+08	TarjeiJS	0.049516	0.072611	31	45	7	9	26430	2657	0
9	3.13E+08	davidllada	0.017678	0.043317	28	34	7	9	34851	4122	1

This result showed that all these six users should belong to the same chess community, even though they were different types of users. Therefore, we will continue to use these six users as the initial core users in the following experiments.

4.2 Sensitivity analysis for the strategy of respecting list for community expansion algorithm.

As mentioned in the code structure part, there are two strategies to set the respecting user list for the community expansion algorithm, which are setting the respecting user list to the refined core users and replace the respecting user list with the previous community. We are going to test which strategy is a more appropriate one. In the experiments, we set the respecting user lost for the core refiner as the initial core users with other hyper parameter the same, and only change the strategy for the community expansion algorithm.

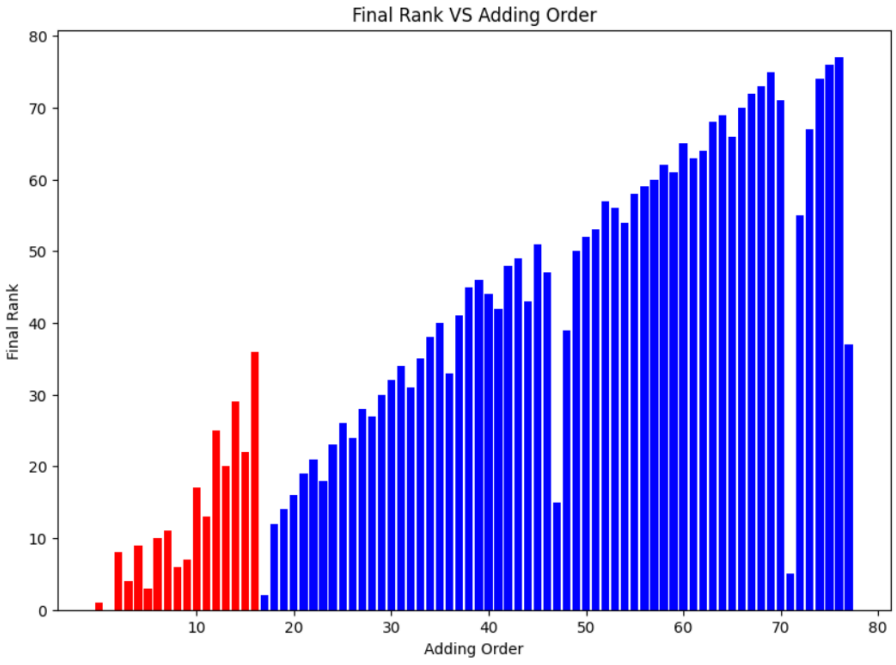


Figure 1. the Final Rank VS Adding Order bar chart using refined core list for community expansion algorithm and initial core list for core refiner.

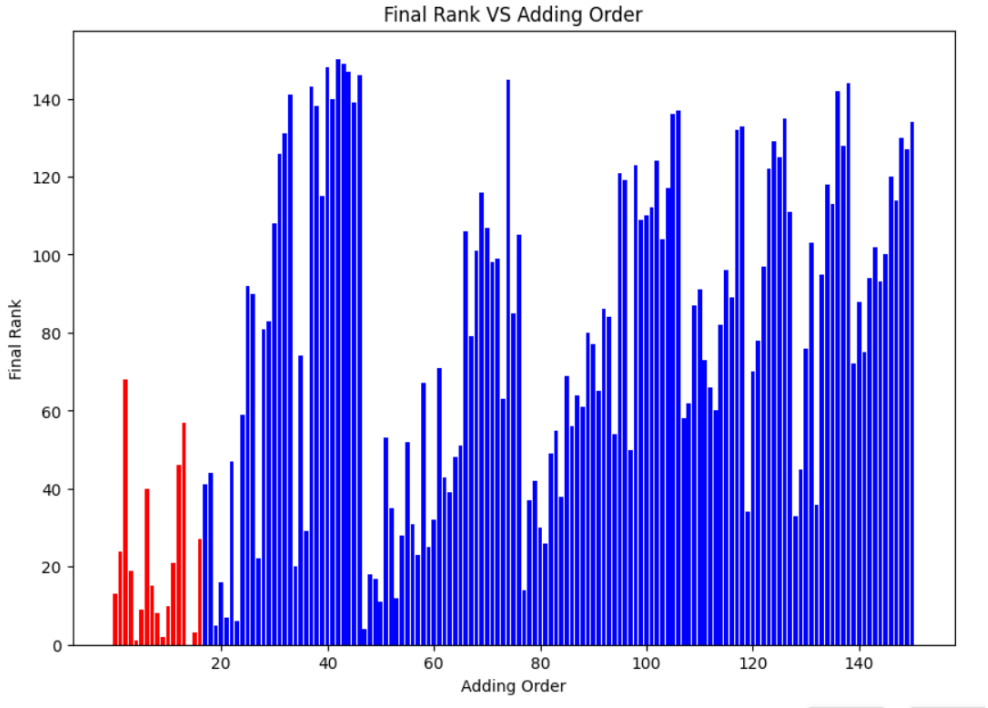


Figure 2. the Final Rank VS Adding Order bar chart using previous community and initial core list for core refiner.

The figure 1 and figure 2 shows the result of the experiments. The x-axis is the adding order of the users in the final expanded community, and the y-axis is the intersection ranking of these users in the final expanded community. The red bars represent the refined core users, while the blue bars represent the new members in the expanded community. The ideal algorithm should result a chart that the later a user was added into the community, the worse rank he/her should have in the final ranking. This is because we expected to add the users with the higher scores at first when expanding the community. Therefore, the strategy that using the refined core list as the respecting user list had a better performance.

4.3 Sensitivity analysis for the strategy of respecting list for core refiner algorithm.

In this experiment, we are going to test which strategy of the respecting list for core refiner algorithm is more appropriate. Based on the result in part 4.2, we set the respecting user list for the community expansion algorithm to the refined core users, with other hyper parameter the same, and only change the strategy of the respecting list for core refiner algorithm.

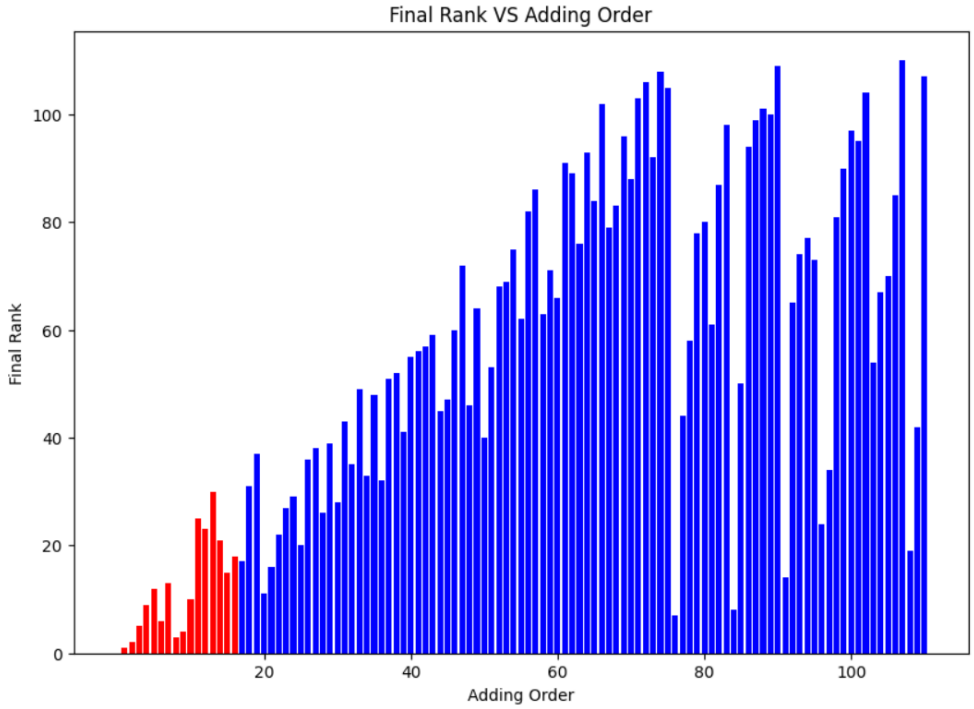


Figure 3. the Final Rank VS Adding Order bar chart using refined core list for community expansion algorithm and previous community for core refiner.

Compared figure 1 and figure 3, when the expanded community size was less than 80, the performance shown in figure 3 is better than figure 1. Meanwhile, although the size of the expanded community in figure 3 is larger than the expanded community in figure 1, the dropping rate of the refined core users in figure 3 was less than the rate in figure 1. Therefore, the result showed that using the previous community as the respecting user list for the core refiner is better than using the initial core users.

Combining the result from the part 4.2 and 4.3, the strategy that using the previous community as the respecting user list for the core refiner and using the refined core users as the respecting user list for the community expansion algorithm has the best performance.

4.4 Sensitivity analysis for the strategy of influence score.

The previous work from Yuchen showed that the community may consist of three types of users:

- **Core Users**: Users that play as hubs in the community. They were expected to have high influence one score, influence two score, production score, and consumption score.
- **Experts**: Users whose tweets and retweets are highly likely to be retweeted by the core. They are likely to have a high influence one score respecting to the core users while likely to have a less influence two score.

- **Consumers**: Users such that a large fraction of their retweets are tweets that are posted or retweeted by the core. They are likely to have a high influence two score respecting to the core users while likely to have a less influence one score.

Since the experiments in the part 4.2 and 4.3 showed the algorithm works better when we respecting to the refined core users in the community expansion algorithm, we can consider these three kinds of users separately in the community expansion algorithm.

Since the refined core users play as the core users in the community, we only need to consider the experts and the consumers. Thus, we change intersection ranking function and the filter candidates strategy (the red code in the above code structure part) as below:

Filter Candidates Strategy

FROM

if *U_Influence*1 ≥ *T_Influence*1 * *Threshold_1*
and *U _Influence*2 ≥ *T_Influence*2 * *Threshold_1*
and *U_Prod* ≥ *T_Prod* * *Threshold_1*
and *U_Con* ≥ *T_Con* * *Threshold_1*
and *U_Follower* < *T_Follower_ceil* * *Threshold_2*
and *U_Follower* ≥ *T_Follower_floor* * *Threshold_3*

TO

if (*U_Influence*1 ≥ *T_Influence*1 * *Threshold_1*
or *U _Influence*2 ≥ *T_Influence*2 * *Threshold_1*)
and *U_Prod* ≥ *T_Prod* * *Threshold_1*
and *U_Con* ≥ *T_Con* * *Threshold_1*
and *U_Follower* < *T_Follower_ceil* * *Threshold_2*
and *U_Follower* ≥ *T_Follower_floor* * *Threshold_3*

Intersection Ranking Strategy

FROM

Intersection ranking ← worst(influence one ranking, influence two ranking, production ranking, consumption ranking)

TO

Intersection ranking ← worst(best(influence one ranking, influence two ranking), production ranking, consumption ranking)

By these changes, we prevent giving an expert with a high influence one score and a low influence two score to a bad rank, as well as ranking a consumer with a low influence one score and a high influence two score to a

bad rank. The result was shown below figure.

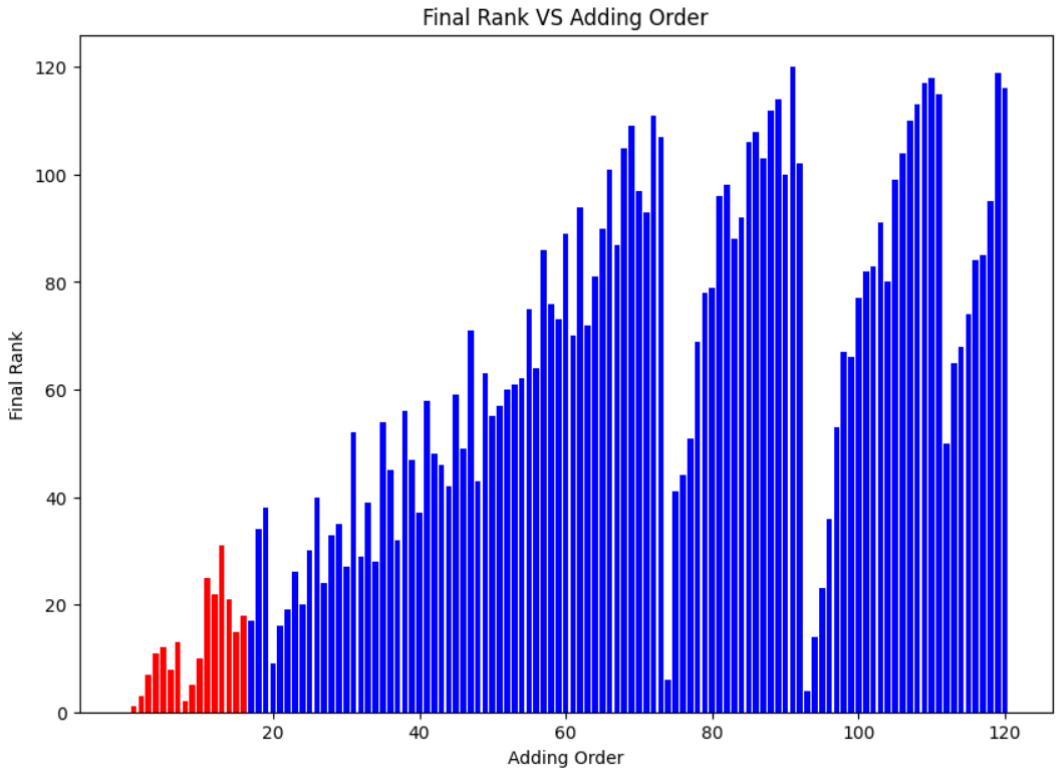


Figure 4. the Final Rank VS Adding Order bar chart combining influence one and influence two, using refined core list for community expansion algorithm and previous community for core refiner.

Compared figure 3 and figure 4, the performance shown in figure is better especially when size of expanded community is larger than 80. Therefore, we would keep the changes for the influence score.

5. Conclusion:

Based on the experiments above, we can conclude that the algorithm which combined influence one and influence two score, used refined core list as the respecting list for community expansion algorithm, and used previous community as the respecting list for core refiner has the best performance. We included the visualization of the full result for the chess community generated by this algorithm in the Appendix part.

6. Future Work:

6.1 Tuning the hyper parameters.

There are several hyper parameters waiting to be tuned.

For core refiner algorithm:

- The threshold for four scores when filtering candidates.

- The Top size when filtering candidates
- The maximum filtered candidates
- The ceiling threshold of global followers when filtering candidates
- The floor threshold of global followers when filtering candidates
- The bottom threshold of local followers when selecting potential candidates.
- The maximum refined core size

For community expansion algorithm:

- The threshold for four scores when filtering candidates.
- The Top size when filtering candidates
- The maximum filtered candidates
- The ceiling threshold of global followers when filtering candidates
- The floor threshold of global followers when filtering candidates
- The bottom threshold of local followers when selecting potential candidates.

6.2 Application in other community.

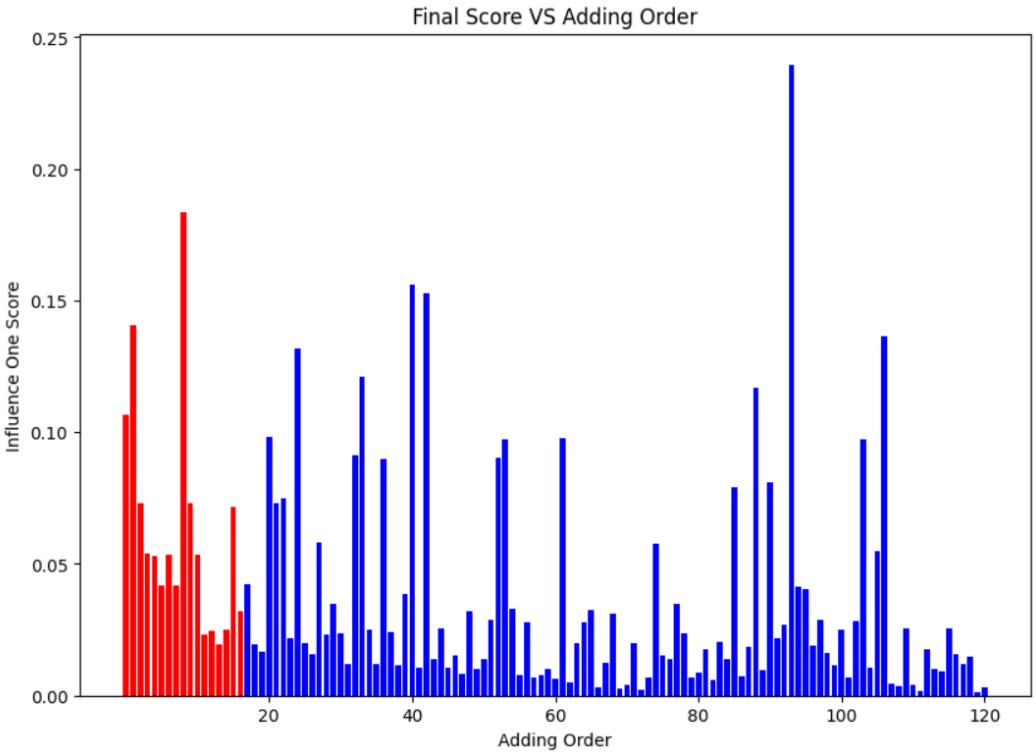
Since the twitter changed its API usage policy, we can't use the API without the academic research twitter developer account. Therefore, the algorithm was only proven the effectiveness in the chess community. In the next step, we should prove the effectiveness of the algorithm in the other community.

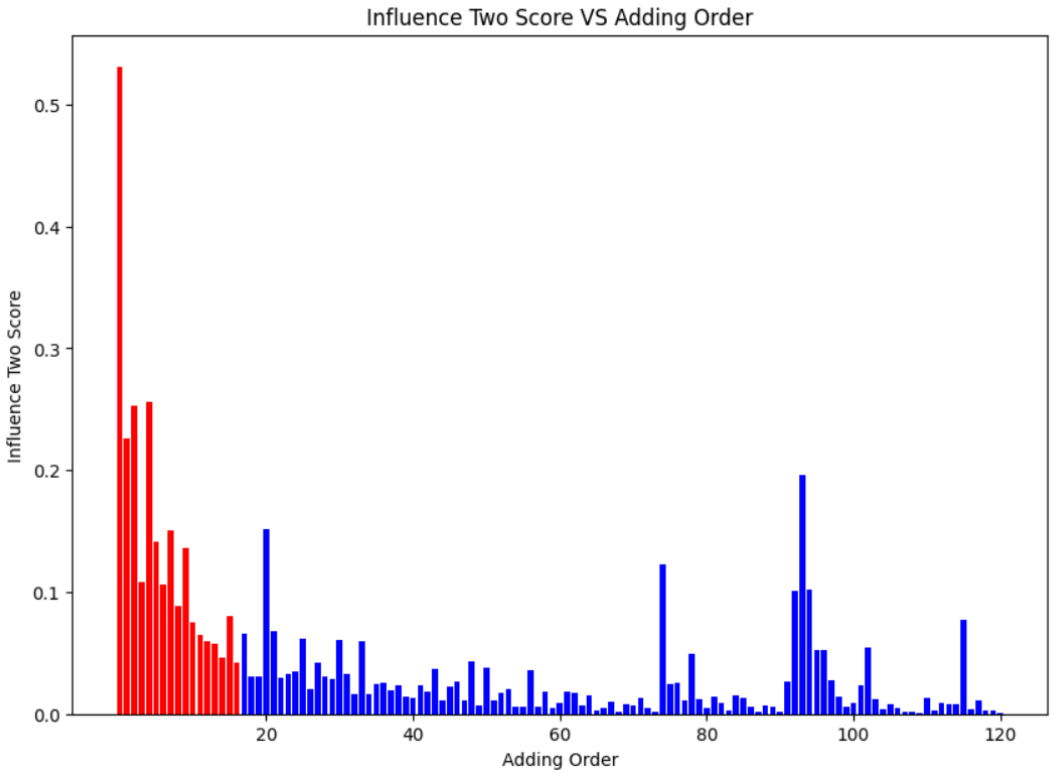
6.3 Address a little issue with the algorithm in the chess community.

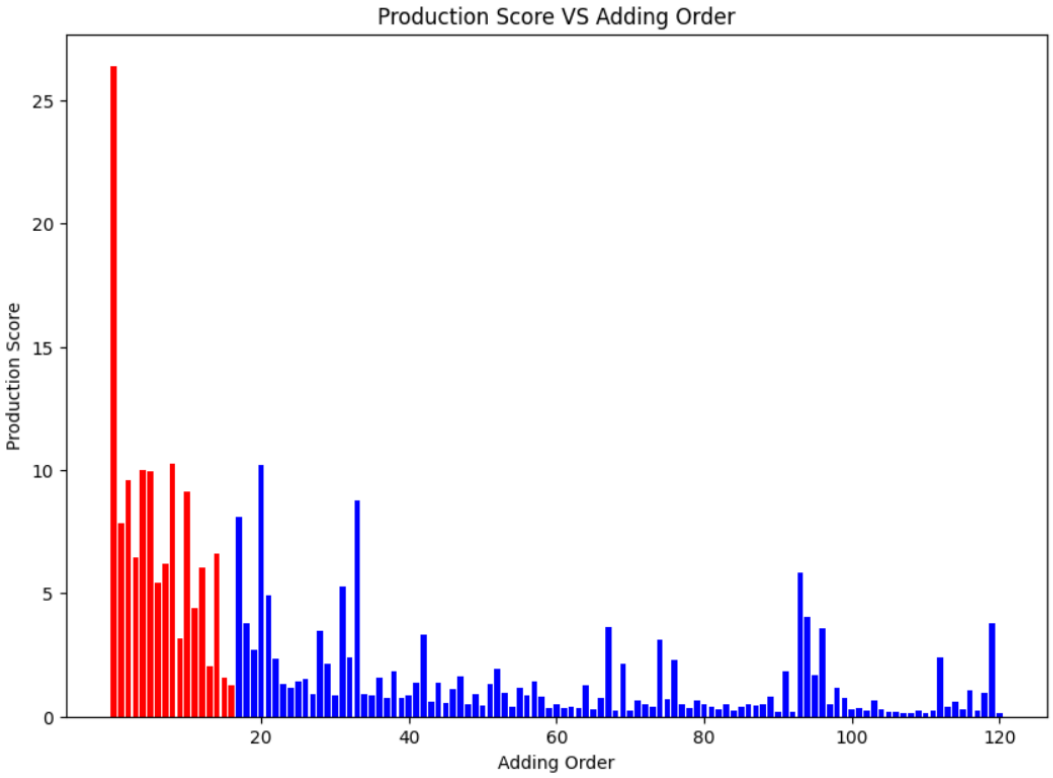
Recall the figure 4, there were still a few users who have a good ranking in the final expanded community but they were added into the community late. It is because in the first iteration they were not selected as the potential candidates. In other words, they were not followed by the refined core users but they have high scores respecting to the refined core users. In the next step, we need to fix this issue of our algorithm. Some possible way includes decreasing the threshold when selecting the potential candidates, modifying the scoring function, and so on.

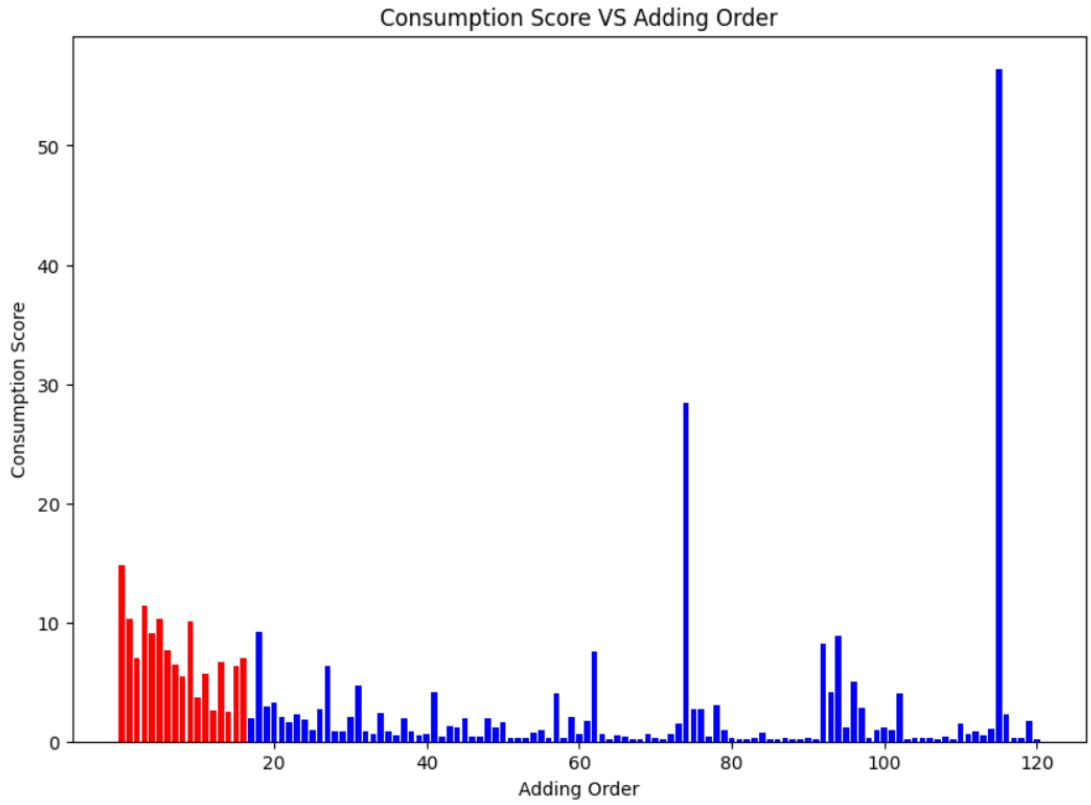
7. Appendix:

The visualization of the full result for the algorithm which combined influence one and influence two, and used refined core list for community expansion algorithm while previous community for core refiner.


Final Score VS Adding Order
Influence One Score
Adding Order
0.25
0.20
0.15
0.10
0.05
0.00
20
40
60
80
100
120


Influence Two Score VS Adding Order
Influence Two Score
Adding Order
0.5
0.4
0.3
0.2
0.1
0.0
20
40
60
80
100
120


Production Score VS Adding Order
Production Score
Adding Order
0
5
10
15
20
25
20
40
60
80
100
120


Consumption Score VS Adding Order
Consumption Score
Adding Order
0
10
20
30
40
50
20
40
60
80
100
120