# Community-based Friend Recommendation on Flick Dataset

Guoxi Liu

## 1 Introduction

Social Networking Service (SNS) is an online platform which people use to build social relations with other people who share similar personal or career interests, activities, backgrounds or real-life connections.

Flickr is an image and video hosting website, web services, and online community. The dataset used in the project contains two parts: users and groups. If there is an edge between two users, it means that these two bloggers are friends; if there is an edge between a user and a group, it means the user is a member of the group. The whole dataset contains 80,513 users and 195 groups. There are 5,899,882 friendship pairs and 107,741 membership pairs. The dataset can be downloaded from ASU Social Computing Data Repository [1].

## 2 Problem

The connections in the social network are not uniformly distributed, which means that the network can be divided into several communities and there are dense connections in the same community but sparse connections between different communities [2]. One problem in the project is to detect the community structure of the Flickr dataset.

Another popular problem in SNS is friend recommendation, which means that the platform should make use of current network structure to help online users finding potential new friends.

## 3 Method

### 3.1 Clustering

The modularity $Q$ of a network is defined as

$$Q = \frac{1}{2m} * \sum_{ij} [A_{i,j} - \frac{k_i * k_j}{2m}] * \delta(C_i, C_j),$$

where $m$ is the total weights of the edges in the network, $k_i$ and $k_j$ are the weights of edges incident to $i$ and $j$ respectively, $\delta(C_i, C_j)$ equals to 1 if $i$ and $j$ are in the same community, and 0 otherwise.

The modularity $Q$ measures the strength of division of a network into modules, and thus an intuitive way for community detection is to maximize this value.

[3] proposed a fast greedy algorithm, that is, we merge two communities that can reach the maximum modularity in each iteration.

[4] introduced a fast algorithm to maximize the modularity by use the concept of modularity gain $\Delta Q$;

$$\Delta Q = \frac{1}{2m}(k_{i,in} - \frac{k_i \sum_{tot}}{m}),$$

where $k_{i,in}$ means the sum of the weights of the links from $i$ to nodes in $C$, and $\sum_{tot}$ denotes the sum of weights of the links incident to nodes in $C$. The algorithm is divided into two phases that are repeated iteratively. In the first phase, each node will be placed into the community of one of its neighbors if the gain is maximum.The second phase is to build a new network whose nodes are the communities found during the first phase.

Label Propagation was another way to detect community structure which was proposed in [5]. In each iteration, we change the community of each node to its neighbor's community which has the maximum sum of edge weights, and stop until there is no change happen.

## 3.2 Recommendation

The practice of friend recommendation is based on the similarity. If two nodes are quite "similar", and there will be a high probability for them to become friends with each other. The similarity used in the project is defined as:

$$Similarity_{(i,j)} = \frac{n_{ij}}{\sqrt{d(i)d(j)}},$$

where $n_{ij}$ denotes the number of common neighbors of node $i$ and $j$, $d(i)$ and $d(j)$ are degrees of node $i$ and $j$ respectively. The similarity is also known as cosine similarity.

Since there are dense connections in the same community, and thus the probability of becoming friends between two users who are in the same community tends to be higher.

## 3.3 Measurements

The new graph is created by randomly removing about 5% edges (about 295,000) from the original graph. This can be done by using `shuf -n NUM edges.txt > test-edges.txt` in the terminal, which means randomly select NUM lines from `edges.txt` and save them to `test-edges.txt`.

The clustering algorithms are evaluated by their run time, reached modularity and number of clusters, whereas the accuracy of recommendation results is calculated as below:

$$Accuracy_{(R)} = \frac{\sum_{i \in N} \delta(R(i), M(i))}{|N|},$$

where $N$ is the user list for friend recommendation, $R(i)$ is the recommendation list for user $i$, $M(i)$ is the missing actual friends of user $i$, and $\delta(A, B) = 1$ if set $A$ and $B$ have at least one common element, and $\delta(A, B) = 0$ otherwise.

# 4 How to Run

The whole project is written in C using igraph library [6]. A `Makefile` is provided in the project root folder and its source code is shown below.

```
# Makefile source code
CC = gcc
CFLAGS = -Wall -g

# project name and header files
PROJECT = flickr
HEADERS = utils.h louvain.h label_propagation.h suggestion.h measurement.h

# path relevant with igraph
IGRAPH_INCLUDE_PATH = /home/guoxil/usr/local/include/igraph
IGRAPH_LIB_PATH = /home/guoxil/usr/local/lib

compile: ${PROJECT}.out

${PROJECT}.out: ${PROJECT}.c ${HEADERS}
	${CC} ${CFLAGS} -o ${PROJECT}.out ${PROJECT}.c -I${IGRAPH_INCLUDE_PATH}
	-L${IGRAPH_LIB_PATH} -ligraph -lm

run: ${PROJECT}.out
	@./${PROJECT}.out

clean:
	rm *.out
```

**Build Steps:**

1. The only things need to be modified in the above `Makefile` are `IGRAPH_INCLUDE_PATH` and `IGRAPH_LIB_PATH`, which are relevant with igraph installation path.

2. After fixing the igraph path issue, just type `make` or `make compile` in the terminal. This will compile the program automatically and then generate an executable file named `flickr.out` if no compilation error occurred.

3. Type `make run` or `./flickr.out` in the terminal to run the program. It runs correctly when it shows "Flickr" and graph information like below in the terminal.

```
______ _ _       _
|  ___| (_)     | |
| |_  | |_  ___| | ___ __
|  _| | | |/ __| |/ / '__|
| |   | | | (__|   <| |
\_|   |_|_|\___|_|\_\_|
```

```
The number of nodes in the whole graph is: 80513.
The number of edges in the whole graph is: 5899882.
The number of nodes in the test graph is: 80513.
The number of edges in the test graph is: 5605000.
```

**Run**

- Below is the main user interface of the program.

```
 \ \        / / | |
  \ \  /\  / /__| | ___ ___  _ __ ___   ___
   \ \/  \/ / _ \ |/ __/ _ \| '_ ` _ \ / _ \
    \  /\  /  __/ | (_| (_) | | | | | |  __/
     \/  \/ \___|_|\___\___/|_| |_| |_|\___|

Please choose the function you want to run:
1. Cluster the test graph using Louvain fast folding.
2. Friends recommendation based on previous clustering results.
3. Measure the quality of prediction.
```

- Run Clustering

  1. Type in `1` and press `enter` in the terminal when displaying the above interface.
  2. Choose the clustering method, where `1` is Louvain fast unfolding method, and `2` is Label Propagation method.
  3. Wait about $4 \sim 20$ seconds (it depends on the selected method), the program will show the number of clusters and save the information to `mycommunities.test`.

- Run Friend Recommendation

  1. Type in `2` and press `enter` in the terminal when displaying the above interface.
  2. Type in the number of randomly selected users for friend recommendation.
  3. Type in the filename to save randomly selected user ids, e.g., `test-users.test`.
  4. Type in the filename to save the recommendation results, e.g., `predictions.test`.
  5. Wait some amount of time (usually 4 minutes for doing recommendation for 1000 users and potential friends are selected only from their community) for the program to finish recommendation.

- Run Prediction Measurement

  1. Type in `3` and press `enter` in the terminal when displaying the above interface.
  2. Type in the filename that contains the user ids, e.g., `test-users.test`.
  3. Type in the filename that contains the recommendation results, e.g., `predictions.test`.
  4. The program will calculate the accuracy based on the provided files.

# 5 Results

## 5.1 Clustering

Figure 1 shows the clustering results in the last step using Louvain fast unfolding method, and Table 1 compares three different clustering algorithms, where the label propagation method runs fastest, and fast greedy uses most time. We can also see that Louvain method has the highest modularity and moderate size of clusters.

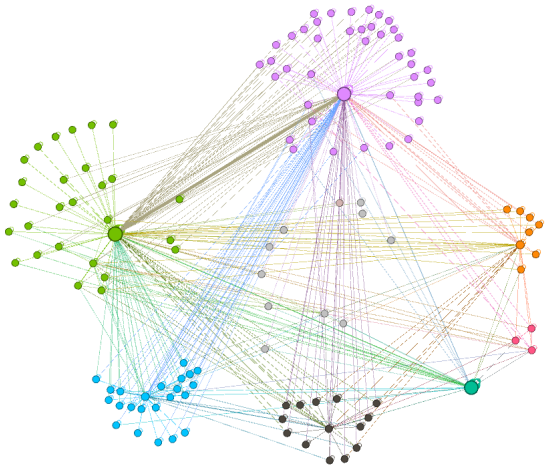

Figure 1: Clustering using Louvain Fast Unfolding

| | Fast Greedy | Louvain Fast Unfolding | Label Propagation |
|---|---|---|---|
| Run Time | 1948.15 seconds | 22.61 seconds | 4.81 seconds |
| Modularity | 0.447 | 0.520 | 0.419 |
| No. of Clusters | 84 | 18 | 6 |

Table 1: Performance of Three Different Clustering Algorithms

## 5.2 Friend Recommendation

To measure the difference between selecting potential users from the whole network and from the same community, first we randomly choose some number of users from the test network, then recommend potential friends for them that have top cosine similarity scores, and finally use the formula mentioned before to compute the accuracy.

Table 2 and 3 show the results when selecting different number of users and returning different number of potential users, where we can obtain much higher accuracy if we only recommend potential friends from the same community.

**Randomly select 100 nodes from the test network:**

| | Top 10 | Top 20 | Top 30 |
|---|---|---|---|
| Whole network | 16% | 18% | 23% |
| Only Community | 27% | 46% | 69% |

Table 2: Performance of Friend Recommendation 1

**Randomly select 1000 nodes from the test network:**

| | Top 10 | Top 20 | Top 30 |
|---|---|---|---|
| Whole network | 17.3% | 21.5% | 23.7% |
| Only Community | 28.2% | 43.4% | 55.4% |

Table 3: Performance of Friend Recommendation 2

# 6 Future Work

The whole Flickr dataset contains not only the user-user friendship but also the user-group membership. However, the project only uses the user graph. So we can make use of the group information to provide better recommendation results in the future, e.g., two users have a high probability if they both join the same group.

# References


[1] R. Zafarani and H. Liu. Social computing data repository at ASU, 2009.

[2] Mark EJ Newman. Detecting community structure in networks. *The European Physical Journal B*, 38(2):321–330, 2004.

[3] Aaron Clauset, Mark EJ Newman, and Cristopher Moore. Finding community structure in very large networks. *Physical review E*, 70(6):066111, 2004.

[4] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment*, 2008(10):P10008, 2008.

[5] Usha Nandini Raghavan, Réka Albert, and Soundar Kumara. Near linear time algorithm to detect community structures in large-scale networks. *Physical review E*, 76(3):036106, 2007.

[6] Gabor Csardi and Tamas Nepusz. The igraph software package for complex network research. *InterJournal, Complex Systems*, 1695(5):1–9, 2006.