Friendship Recommendation Modifications

DE-XUAN HUANG, GUAN-TING CHEN, XIAO-QING LIN, YONG-LUN TAO

Abstract—**In this study, we propose an enhancement to traditional friend recommendation systems by developing a novel system that allows users to modify their profile characteristics to align more closely with target individuals. This adaptive approach aims to increase the relevance and accuracy of friend suggestions. We conducted a series of experiments to evaluate the performance of our model against conventional recommendation systems. The results demonstrate that our modified system not only improves the precision of recommendations but also enhances user satisfaction by providing more personalized and meaningful connections.**

Keywords—social network, friendship recommendation, link predicition

I. Introduction

Social media has revolutionized the way people build and maintain social relationships, connecting individuals who share similar backgrounds, values, and real-world connections. Many users rely on social media platforms to expand their social circles and foster new connections. A critical component of these platforms is the friend recommendation system, which plays a pivotal role in attracting and retaining users.

Traditionally, friend recommendation has been implemented using graph-based heuristics. However, recent advancements in Graph Neural Networks (GNNs) have introduced more sophisticated methods for this task. GNNs are capable of capturing intricate similarities between users by leveraging the underlying graph structure of social networks.

In this project, we enhance the traditional friend recommendation system by allowing users to modify their profile characteristics to better align with target individuals. This novel approach aims to provide more personalized and relevant friend suggestions, thereby improving the overall user experience.

To achieve this, we utilize the Facebook social circles dataset, which includes detailed node features (profiles), circles, and ego networks. This dataset allows us to analyze the impact of profile modifications on friend recommendations within a well-defined social graph.

II. Related work

We reference some work from existing research, particularly in the field of Graph Neural Networks (GNNs). Specifically, we utilize methodologies and insights from Graph Convolutional Networks (GCNs) and GraphSAGE.

Graph Convolutional Networks (GCNs) are a type of neural network specifically designed to handle graph-structured data. GCNs extend the concept of convolution from traditional grid data (like images) to graph data, enabling the model to capture local graph structure and node features effectively. This is achieved through a series of convolutional layers that aggregate information from a node's neighbors, thereby learning a node representation that incorporates both its features and the structure of the graph. GCNs have been widely used in various applications, including node classification, link prediction, and graph classification, due to their ability to generalize across different types of graphs.

GraphSAGE (Graph Sample and AggregatE) is another prominent framework in the GNN domain, designed to efficiently generate node embeddings for large graphs. Unlike GCNs, which operate on the entire graph at once, GraphSAGE samples a fixed-size neighborhood for each node and then aggregates information from this sampled neighborhood. This approach not only makes GraphSAGE scalable to large graphs but also allows for inductive learning, where the model can generate embeddings for previously unseen nodes. The aggregation function in GraphSAGE can be mean, LSTM, or pooling, each providing different ways of summarizing the information from a node's neighbors.

We use GCNs and GraphSAGE mainly for friend recommendation. We use these two to do cross comparison with each exploiting method, and further refine our algorithm.

III. Methodology

In our project, we aim to provide user a way to improve their chances of getting a specific targeted new friend. Our experiment is mainly conducted a friend ranking system. The ranking system leverages several component.

*A. **Gnn***

Graph Neural Networks (GNNs) are specialized neural networks that operate directly on graph structures. They generate node embeddings through multiple layers of non-linear transformations, which are informed by the graph's structure. This enables GNNs to perform various tasks, including node classification, link prediction, community detection, and network similarity. For friend recommendation, we focus on the link prediction task, which involves estimating the likelihood of connections (friendships) between nodes (users).

GNNs generate node embeddings by aggregating information from local network neighborhoods through message-passing layers. Each layer updates node embeddings based on the embeddings of neighboring nodes, allowing the model to capture complex patterns in the graph.

*B. **GraphSAGE***

GraphSAGE is an inductive framework for learning node embeddings on large graphs. It generates low-dimensional vector representations for nodes, making it suitable for graphs with rich attribute information. Unlike transductive GNN models like Graph Convolutional Networks (GCN), GraphSAGE can generalize to unseen nodes, making it ideal for dynamic social networks where user connections change over time.

In this project, we implement both GraphSAGE and GCN using the DGL library to compare their performance on the friend recommendation task.

*C. **Link Prediction***

To transform node embeddings into friend recommendations, we model the link prediction task by estimating a likelihood score for potential edges (friendships) in the graph. This score is calculated using the dot product of the embeddings of two nodes. By sorting these scores, we identify the top K potential friends for each user.

D. *Rank Potential Friends*

After calculating the likelihood scores, we generate a list of friend recommendations for each user. Instead of recommending the users with the absolute highest scores, we select a random subset from the top-scoring users. This approach ensures a dynamic and diverse set of friend suggestions, enhancing the user experience.

To train and refine the ranking system, we utilize the Facebook social circles dataset, which includes detailed node features (profiles), circles, and ego networks. This dataset allows us to analyze the impact of profile modifications on friend recommendations within a well-defined social graph.

E. *Dataset Description*

The dataset used in this project consists of Facebook social circles data collected from survey participants using the Facebook App. It includes node features, circles, and ego networks. An "ego" network refers to the central user and their connections. In this project, we focus on a single ego network, comprising 347 nodes and 5038 edges. Each node is represented by a 224-dimensional feature vector.

In summary, this project presents a novel friend recommendation system that allows users to modify their profile characteristics to better match target individuals. By leveraging the Facebook social circles dataset and implementing state-of-the-art GNN models, we demonstrate the potential of our system to provide more accurate and personalized friend recommendations.

IV. Experiments

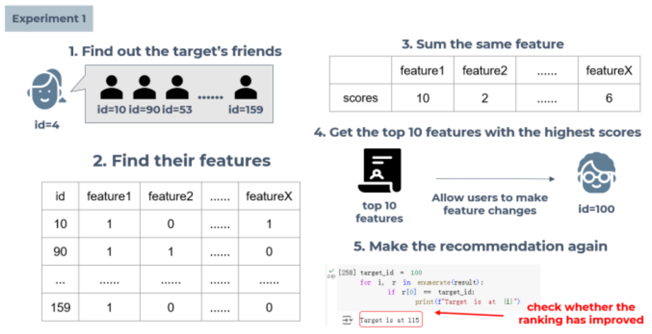
The algorithm mentioned for recommending a list of friends first utilizes the existing social network and the features of each user within that network as inputs. It employs Graph Neural Networks (GNN) and GraphSAGE to obtain embeddings for each user. By relying on the edge likelihood of each node, it performs Link Prediction, ultimately producing a list of recommended friends. We will use the list generated by this algorithm for validation purposes, adjusting different input values to guide the results towards the expected ranking.

A. *Experiment 1: Modifying Features Based on the Target's Existing Friends*

Our idea is to provide the user with 10 recommended features to modify. By adjusting these features, the user can improve their ranking in the target's recommended friend list. To determine these 10 features, we will first obtain the target's current friend list and extract the features of all nodes on that list (where a feature is present it is marked as 1, and if absent it is marked as 0). We will then sum the values and select the top ten features with the highest total scores as the basis for the user's modifications. An illustration of this process is shown in Fig 1.

B. *Experiment 2: Modifying Features Based on the Target's Recommended Friends*

Similarly, we will provide the user with 10 recommended features to modify. By adjusting these features, the user can improve their ranking in the target's recommended friend list. The difference in this experiment is that to determine these 10 features, we will first obtain the target's recommended friend list generated by the friend recommendation algorithm. We will extract the features of the top 10 nodes on that list (where a feature is present it is marked as 1, and if absent it is marked as 0). We will then sum the values and select the top ten features with the highest total scores as the basis for the user's modifications. An illustration of this process is shown in Fig 2.

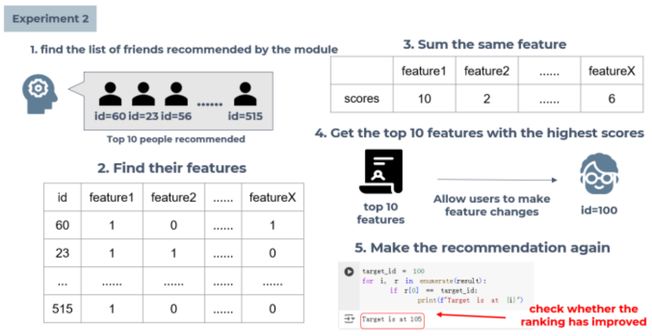

Experiment 1
1. Find out the target's friends
id=10 id=90 id=53 .... id=159
id=4
2. Find their features
3. Sum the same feature
4. Get the top 10 features with the highest scores
top 10 features
Allow users to make feature changes
id=100
5. Make the recommendation again
check whether the ranking has improved

id	feature1	feature2	......	featureX
10	1	0	......	1
90	1	1	......	0
...	......	......	......	......
159	1	0	......	0

	feature1	feature2	......	featureX
scores	10	2	......	6

```
[258] target_id = 100
      for i, r in enumerate(result):
          if r[0] == target_id:
              print(f"Target is at {i}")
Target is at 115
```

Fig 1. : Experiment 1 pipeline


Experiment 2
1. find the list of friends recommended by the module
id=60 id=23 id=56 .... id=515
Top 10 people recommended
2. Find their features
3. Sum the same feature
4. Get the top 10 features with the highest scores
top 10 features
Allow users to make feature changes
id=100
5. Make the recommendation again
check whether the ranking has improved

id	feature1	feature2	......	featureX
60	1	0	......	1
23	1	1	......	0
...	......	......	......	......
515	1	0	......	0

	feature1	feature2	......	featureX
scores	10	2	......	6

```
target_id = 100
for i, r in enumerate(result):
    if r[0] == target_id:
        print(f"Target is at {i}")
Target is at 105
```

Fig 2. : Experiment 2 pipeline

C. *Experiment 3: Making Friends with Target's Key Neighbors by Random Walk with Restart*

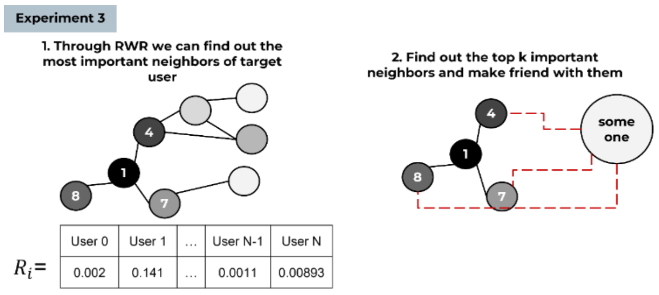
The above two experiments focused on modifying the user features to approach the target user. However, the GNN-based method will also consider the user-user friendships to predict the recommended scores. Thus, it is also important to modify the user's friendships. To achieve this, we first adopt the random walk with restart (RWR) algorithm to determine which users are the most important neighbors. The RWR algorithm is formulated as the formula 1 and if it runs almost 50 iterations, the final score R_i will converge. Through RWR, it generates the score of all users in the graph and gives the closer neighbor the higher score.

$$R_{i+1} = (1-\alpha)W_{n\times n}R_i + \alpha E \quad (1)$$

Furthermore, the original RWR algorithm is applied to the normalized adjacency matrix. However, it assigns the same weight to every neighbor during transitions. In addition to using the simple adjacency matrix, we also construct a feature-based normalized adjacency matrix, which is based on the dot product of any two users. We provide a simple example using only four users, and the result is shown in Fig 4.

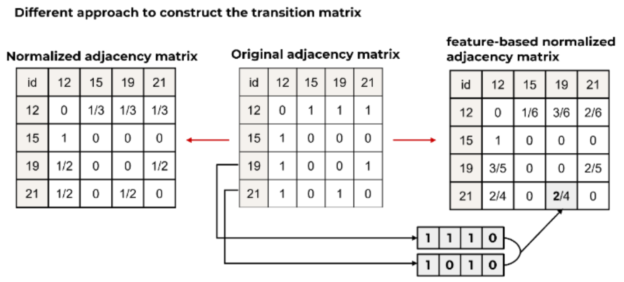
D. *Experiment 4: Making Friends and Modifying Features with Simulated Annealing*

All the above experiments only do one kind of adjustment at a time. To adjust both user features and user relationships, we adopt the simulated annealing algorithm to find the best adjustments. Simulated Annealing is more likely to accept the non-optimal solution at earlier iterations, and it tends to accept the better solution at the later iterations. This method allows it to escape local minima and find better solutions.


Experiment 3
1. Through RWR we can find out the most important neighbors of target user
2. Find out the top k important neighbors and make friend with them
4
1
8
7
some one

	User 0	User 1	…	User N-1	User N
$R_i=$	0.002	0.141	…	0.0011	0.00893

Fig 3. : After RWR, each user will be assigned a score, and our approach is to make friends with the top k users of the highest score.


Different approach to construct the transition matrix
1 1 1 0
1 0 1 0

Normalized adjacency matrix

id	12	15	19	21
12	0	1/3	1/3	1/3
15	1	0	0	0
19	1/2	0	0	1/2
21	1/2	0	1/2	0

Original adjacency matrix

id	12	15	19	21
12	0	1	1	1
15	1	0	0	0
19	1	0	0	1
21	1	0	1	0

feature-based normalized adjacency matrix

id	12	15	19	21
12	0	1/6	3/6	2/6
15	1	0	0	0
19	3/5	0	0	2/5
21	2/4	0	**2/4**	0

Fig 4. : An example of 4 users' adjacency matrix.

To leverage it on our task, we first define the solution as a k-sized vector with each value representing either changing the current relationship of the user or altering the user's feature. During each iteration, we will randomly pick a position in the old solution and randomly alter its value, as shown in Fig 5.

Besides, we also need a criterion to judge how well our approach is. We simply leverage the RWR on the feature-based normalized adjacency matrix approach from Experiment 3, and check whether the score improves or not after modification. If the score improves, then we replace the old solution to the new one. Otherwise, simulated annealing will decide to accept it or reject it based on its probability of acceptance. The whole framework is demonstrated in Fig 6.

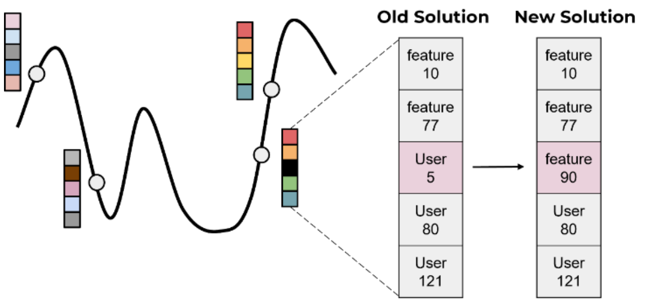


Fig 5. : The demonstration of the modification from old solution to new one.

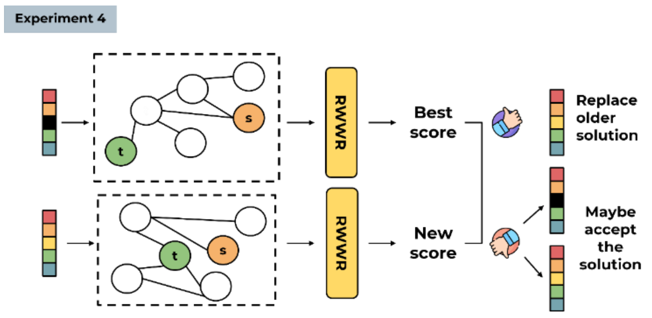
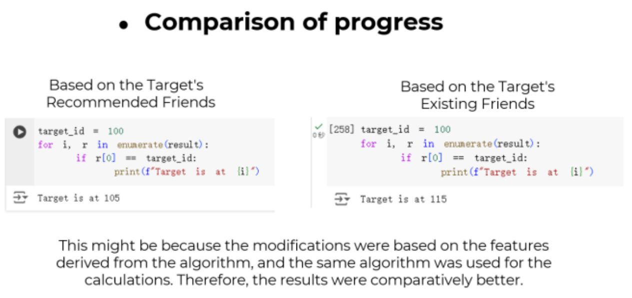


Fig 6. : The whole framework of simulated annealing approach.

V. Conclusions

From the experiments conducted, we can infer that correctly altering the input data can indeed guide the ranking results towards the user's expected outcomes. However, the effectiveness varies depending on the nature of the adjustments made. Experiments 1 and 2 show that different feature modification methods can have varying impacts on the ranking results, as illustrated in Fig 7. The method used in Experiment 2 was specifically designed for the algorithm used in this study, so it may not be applicable in other cases.



- **Comparison of progress**

Based on the Target's Recommended Friends

```
target_id = 100
for i, r in enumerate(result):
        if r[0] == target_id:
                print(f"Target is at {i}")
```

```
Target is at 105
```

Based on the Target's Existing Friends

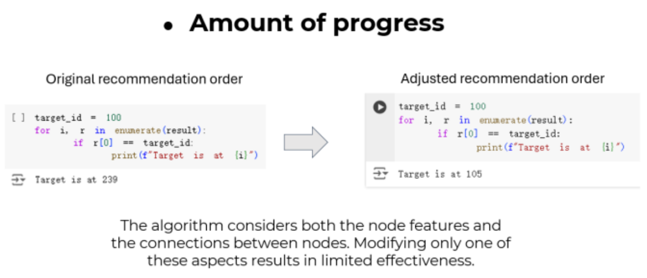
```
[258] target_id = 100
      for i, r in enumerate(result):
              if r[0] == target_id:
                      print(f"Target is at {i}")
```

```
Target is at 115
```

This might be because the modifications were based on the features derived from the algorithm, and the same algorithm was used for the calculations. Therefore, the results were comparatively better.

Fig 7. : Comparative analysis of different feature extraction methods

Additionally, the improvement in rankings was not significant, as shown in Fig 8. This may be because the algorithm used in this study not only considers the feature values of the nodes for friend recommendations but also takes into account the current state of the social network to derive the embeddings for each node. Therefore, it would be more appropriate to consider modifying both the feature values and the social network context to achieve better results.



- **Amount of progress**

Original recommendation order

```
[ ] target_id = 100
    for i, r in enumerate(result):
            if r[0] == target_id:
                    print(f"Target is at {i}")
```

```
Target is at 239
```

Adjusted recommendation order

```
target_id = 100
for i, r in enumerate(result):
        if r[0] == target_id:
                print(f"Target is at {i}")
```

```
Target is at 105
```

The algorithm considers both the node features and the connections between nodes. Modifying only one of these aspects results in limited effectiveness.

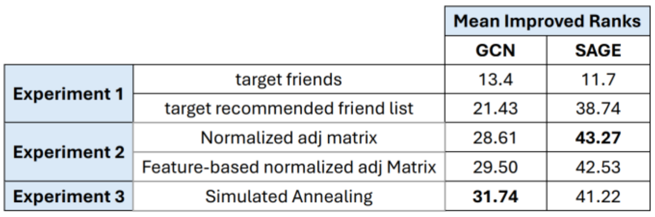
Fig 8. : Analysis of progress range results

To compare all experiment, we randomly select users in rank list from 100 to 200 as target users, and we simply pick out 100 samples to run the whole comparison. We adopt the rank improvement to judge the performance, which is just the subtraction of rank before and after modification. Furthermore, we also compare the performance of different link prediction models, i.e. GCN and GraphSAGE. We show the results in the Table 1.

We conclude the results from the table in two points. (1) Almost all of our experiments achieve better scores with GraphSAGE compared to GCN. We assume this is because GraphSAGE is trained on randomly sampled neighbors, which is similar to our RWR approach. (2) Experiments 2 and 3 achieve higher scores than Experiment 1. We conclude that altering the graph structure is more effective than changing user features. This outcome is intuitive since it directly makes the source user one of the neighbors of the target user.

Additionally, we aim to determine if there is any relationship between the rank range in the ranking list and the improved rank value. We select four intervals: (50-100), (100-150), (150-200), and (200-250), and conduct the same comparison settings. The results are shown in Fig 9. It is clear that when our source user is in the earlier part of the target user's ranking list, it is difficult to improve the rank through our adjustments. On the other hand, the rank can improve significantly when it is in the latter part of the ranking list. This is because the earlier part of the ranking list means that the source user is already a neighbor of the target user, and their features are already similar to each other.

All the above experimental results can be reproduced using the following GitHub links. (https://github.com/larrychen20011120/sna-make-friend)



		Mean Improved Ranks	
		GCN	**SAGE**
Experiment 1	target friends	13.4	11.7
	target recommended friend list	21.43	38.74
Experiment 2	Normalized adj matrix	28.61	**43.27**
	Feature-based normalized adj Matrix	29.50	42.53
Experiment 3	Simulated Annealing	**31.74**	41.22

Table 1. : The Whole Comparison of three experiments.

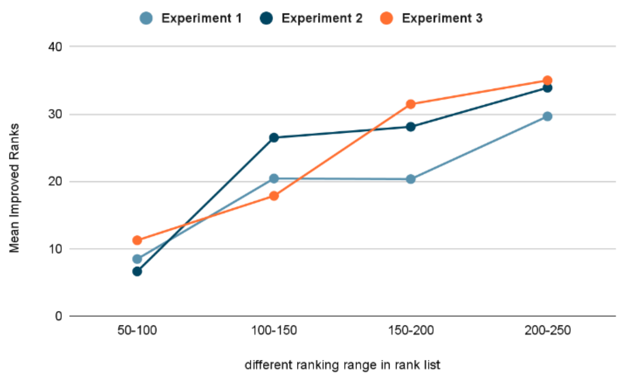


Fig 9. : Different ranking range and its performance.

References

[1] J. McAuley and J. Leskovec. Learning to Discover Social Circles in Ego Networks. NIPS, 2012.

[2] Wang, Y. (2022, February 9). Friend Recommendation Using GraphSAGE. Medium. https://medium.com/stanford-cs224w/friend-recommendation-using-graphsage-ffcda2aaf8d6

[3] Bertsimas, D., & Tsitsiklis, J. (1993). Simulated annealing. *Statistical science*, *8*(1), 10-15.

[4] Tong, H., Faloutsos, C., & Pan, J. Y. (2008). Random walk with restart: fast solutions and applications. *Knowledge and Information Systems*, *14*, 327-346.

[5] Kumar, A., Singh, S. S., Singh, K., & Biswas, B. (2020). Link prediction techniques, applications, and performance: A survey. *Physica A: Statistical Mechanics and its Applications*, *553*, 124289

Self-Assessment

Member Name	Member Contribution
黃德軒	負責提供研究方向、實驗實施、結論探討、報告內容彙整
陳冠廷	負責提供研究方向、實驗實施、結論探討
林曉慶	提議研究方向、部分報告內容撰寫
陶泳綺	專案討論、相關研究、研究方法、課堂報告、書面報告