Graph Attention Network-Based Recommendation of Issue Resolution Participants in Open Source Communities: Postprint

Authors: Zhao Haiyan, Xia Wenzhong, Cao Jian, Chen Qingkui

Date: 2022-04-07T15:01:57+00:00

Abstract

In open source communities, the promptness and quality of responses to developers' issues critically determine community vitality. Therefore, identifying and recommending appropriate problem solvers for newly submitted issues facilitates community development. This paper proposes a problem solver recommendation model based on a dual-layer graph attention network (GAT-UCG), constructed from records of developer collaboration relationships and developer issue participation. Specifically, we first extract information regarding issue participants and developer interactions to construct a developer-issue participation graph and a developer collaboration graph, respectively. An attention mechanism is then employed to reallocate edge weights, and Top-N problem solver recommendations are generated based on issue node embedding representations obtained from the output layer. Experiments conducted on 7,352 issues from popular GitHub repositories demonstrate that the proposed GAT-UCG model achieves superior performance over baseline methods across three metrics: recommendation accuracy, recall, and F-Score.

Full Text

Preamble

Vol. 39 No. 8
Application Research of Computers
Accepted Paper

Graph Attention Network Based Participant Recommendation for Issue Resolution in Open Source Community

Zhao Haiyan[1],[2],[3], Xia Wenzong[1],[2],[3]†, Cao Jian[4], Chen Qingkui[1],[2],[3]

[1] Shanghai Key Lab of Modern Optical System, Shanghai 200093, China
[2] Engineering Research Center of Optical Instrument & System, Ministry of Education, Shanghai 200093, China
[3] School of Optical-Electrical & Computer Engineering, University of Shanghai for Science & Technology, Shanghai 200093, China
[4] Dept. of Computer Science & Technology, Shanghai Jiao Tong University, Shanghai 200030, China

Abstract: In open source communities, the speed and quality of responses to developer-submitted issues determine community vitality. Therefore, identifying and recommending suitable participants for newly submitted issues facilitates community development. This paper proposes a two-layer Graph Attention Network model for issue resolution participant recommendation (GAT-UCG) based on developers' collaborative relationship records and historical issue participation. The method first obtains participant information and developer interaction data to construct both a developer-issue participation graph and a developer collaboration graph. Attention mechanisms then redistribute edge weights, and the output layer generates issue node embeddings for Top-N participant recommendations. Experiments on 7,352 issues from popular GitHub repositories demonstrate that GAT-UCG outperforms baseline methods in recommendation accuracy, recall, and F-Score.

Keywords: recommendation system; issue tracking; graph attention network; participant recommendation; comment network

0 Introduction

Modern open source software development relies on communities that provide efficient collaboration platforms. GitHub, as the world's largest open source platform, attracts developers globally. To facilitate project communication, anyone can create issues, which may be addressed by core team members or external contributors interested in the problem. Numerous issues await developer responses and resolution [1]. Applying recommendation technology to identify suitable participants for each issue can accelerate resolution speed and quality, thereby promoting community collaboration and development [2,3].

Recent research has proposed various approaches that recommend participants based on developer profiles, historical problem-solving characteristics, and issue features [4,5]. Zhou et al. employed Structural Equation Modeling (SEM) to analyze open source communities and found that social identity is the primary factor determining user knowledge contribution [6]. Morris et al. discovered that many issues are resolved by closely connected friends, and relationship strength influences developer participation enthusiasm [7,8]. Incorporating developer collaboration relationships into recommendation models can enhance knowledge contribution willingness and improve problem-solving efficiency [9,10]. However, existing models primarily rely on developer expertise and consider only direct social relationships [11,12], failing to capture the potential and complex effects of social relationships on participation motivation. This paper designs a graph attention network-based participant recommendation model that introduces a user collaboration graph to improve recommendation effectiveness.

The main contributions are: (1) Based on open source community collaboration characteristics, we combine developer collaboration information and issue participation records with graph attention networks to construct a two-layer GAT-UCG model comprising a developer collaboration graph and a developer-issue participation graph. (2) Experiments on 7,352 issues from popular GitHub repositories [13] demonstrate that GAT-UCG outperforms baseline methods in accuracy, recall, and F-Score.

Collaborative relationships among developers play crucial roles in issue resolution. To fully exploit developer collaboration networks, we design a two-layer Graph Attention Network model (GAT-UCG) based on developer comment interactions and issue participation records. The overall architecture is shown in Figure 1.

1 Related Work

Numerous studies have addressed issue participant recommendation. Jiang et al. proposed a multi-attribute-based reviewer recommendation method incorporating developer activity and text similarity, finding activity to be the most important attribute [14]. Chen et al. designed an answerer recommendation system that targets developers with relevant expertise, noting that timely issue processing promotes interaction and encourages answerers to improve their responses [15]. Davoodi et al. presented a hybrid expert recommendation system using social network-based collaborative filtering to improve prediction accuracy [16]. Xu proposed a novel social network recommendation method using matrix factorization to alleviate sparsity and improve accuracy and diversity in complex contexts [17]. Liu et al. developed a hybrid recommendation algorithm combining information indexing, comment networks, and entropy methods, which outperformed individual approaches [1].

Traditional recommendation systems suffer from poor generalization, insufficient expressive power, and difficulty handling non-Euclidean data [18-20]. Graph Neural Networks (GNNs) have achieved remarkable success in mining hidden features from complex networks [21]. For GNN-based recommendation, Kipf et al. proposed a semi-supervised Graph Convolutional Network (GCN) for label classification using graph topology and node information [22]. Zhang et al. introduced a heuristic link prediction method based on GNNs that learns heuristics from local subgraphs while maintaining strong generalization [23]. Zhang et al. also proposed a GCN-based user representation learning method for recommendation using multi-layer graph convolutions [24].

While GCN effectively represents node features, its transductive nature is unsuitable for rapidly iterating open source software. To improve scalability, Hamilton et al. proposed GraphSAGE, which enhances flexibility and generalization [25]. However, different developers contribute differently to issues, requiring importance weighting of edges between developer nodes. Veličković et al. introduced Graph Attention Networks (GAT) that assign different weights to neighbors based on their features [26]. Subsequent improvements include Tao et al.' s multimodal GAT recommendation algorithm [27], Fan et al.' s GraphRec architecture for social recommendation [28], Guo et al.' s GNN-SoR framework that abstracts user and item features into two graphs [29], and Wang et al.' s use of clustering functions with attention mechanisms for user-item representation [30].

Our model first constructs developer features from comment data in issue resolution processes, then uses graph attention to learn neighbor weights and propagate features. Issue tags directly reflect the topic and module of questions, so we use One-Hot encoding of issue tags as node features for participant recommendation. Compared to existing models, ours more comprehensively integrates collaboration relationships.

2 Graph Attention Network Based Participant Recommendation

We first analyze comment data in open source communities to validate the importance of social relationships in issue resolution.

2.1 Importance of Social Relationships for Developer Participation

To verify the proportion of historical collaborations in new issue discussions, we analyzed data from two popular GitHub repositories: TensorFlow and Kubernetes. We selected issues within specific time periods, established monthly social interaction baselines using the first month' s data, and compared subsequent months' interactions against this baseline to calculate the proportion of historically interacted relationships. We define historical interaction as users who have commented in the same issue. Results are shown in Table 1.

Table 1. Statistics of Social Relationships in TensorFlow and Kubernetes

Time Bucket	Interaction	Historical Interaction	Historical Proportion
TensorFlow			
3/22/2020-4/30/2020	-	-	21.61%
4/30/2020-5/31/2020	-	-	25.50%
5/31/2020-6/30/2020	-	-	29.71%

Time Bucket	Interaction	Historical Interaction	Historical Proportion
6/30/2020-7/31/2020	-	-	49.89%
12/28/2020-1/31/2021	-	-	57.69%
1/31/2021-2/28/2021	-	-	62.06%
2/28/2021-3/31/2021	-	-	64.14%
Kubernetes			
3/31/2021-4/30/2021	-	-	69.47%
4/30/2021-5/31/2021	-	-	67.60%
5/31/2021-6/30/2021	-	-	61.80%

The results show that in Kubernetes, historically interacted relationships account for 61.80% of monthly interactions on average, with similar patterns in TensorFlow. This demonstrates that developers frequently collaborate with previously interacted partners in new issue discussions, and this proportion steadily increases over time as collaboration networks expand. Incorporating historical collaboration information thus improves recommendation performance.

2.2 Multi-Head Graph Attention Layer

The multi-head graph attention layer serves as GAT-UCG' s basic unit, with architecture shown in Figure 2. Input is a set of node features $\{p_1, p_2, p_3, ..., p_N\} \in \mathbb{R}^{F \times N}$, where N is the number of nodes and F is the feature dimension per node. Each layer outputs new node features $\{p'_1, p'_2, p'_3, ..., p'_N\} \in \mathbb{R}^{F' \times N}$. To transform input features into high-order embeddings, we introduce self-attention mechanisms in each layer, with calculations shown in Equations (1) and (2).

The attention layer's output embedding is calculated as in Equation (3), where σ is a non-linear activation function, G is the input graph, and $\mathcal{N}_i$ represents node i' s first-order neighbors. To obtain stable self-attention results, we employ K independent attention heads, concatenating their outputs. In the final layer, we replace concatenation with averaging to reduce output dimensionality, as shown in Equations (4) and (5), where $\|$ denotes concatenation, K is the number of heads, α_{ij}^k is the attention coefficient from head k, and W^k is the corresponding linear transformation weight matrix. Multi-head attention distributes attention across relevant features between center and neighbor nodes, enhancing model capacity.

2.3 Weight Propagation in Developer Collaboration Graph

The developer collaboration graph layer handles weight propagation between developers. We define the collaboration graph as a weighted directed graph $\mathcal{G}_D = (\mathcal{V}_D, \mathcal{E}_D, W)$, where $\mathcal{V}_D$ represents the developer node set and $\mathcal{E}_D$ represents edges. An edge e_{ij} exists from developer D_i to D_j if D_i has commented on or referenced D_j's issue or reply. We extract @mentions and comment references using regular expressions to obtain collaboration relationships. The weight set W reflects edge importance, learned through attention mechanisms, with α_{ij} representing the weight between D_i and D_j. Developer node embeddings are obtained via Multi-hot encoding of historical issue tags they participated in. The output of the developer collaboration graph attention mechanism is shown in Equation (6), where p_{D_i} and p_{D_j} are developer node embeddings and $\mathcal{N}_{D_i}$ represents D_i' s first-order neighbors.

Figure 3 shows a partial example of the developer collaboration graph, where edges exist between developers with collaboration relationships (e.g., D_4 to D_5 with weight α_{45}).

2.4 Weight Propagation in Developer-Issue Participation Graph

We define the developer-issue participation graph as a bipartite graph $\mathcal{G}_{ID} = (\mathcal{V}_I \cup \mathcal{V}_D, \mathcal{E}_{ID}, W)$, where $\mathcal{V}_I$ is the issue set and $\mathcal{V}_D$ is the developer set. Edge e_{ij} exists between issue node I_i and developer node D_j if developer D_j commented on issue I_i at least once. The weight set W reflects edge importance, learned via attention mechanisms, with α_{ij} representing the weight between I_i and D_j. The output of the developer-issue participation graph attention mechanism is shown in Equation (7), where p_{I_i} and p_{D_j} are issue and developer embeddings, and $\mathcal{N}_{I_i}$ represents issue node I_i' s first-order neighbors.

Figure 4 illustrates a partial developer-issue participation graph example, showing edges between developers and issues they participated in (e.g., $\alpha_{1I_1}, \alpha_{2I_1}, \alpha_{3I_1}$).

2.5 Model Prediction and Optimization

To learn model parameters and better capture issue-developer features, we use the LogSoftmax function for participant prediction, as shown in Equation (8), where a_i is the probability that issue node I should be assigned to developer D_i. This yields probabilities for assigning issue I to each developer, enabling Top-N recommendations based on sorted probabilities. LogSoftmax accelerates computation and improves stability. Model parameters are updated using gradient descent.

3 Experiments

3.1 Issue and Comment Data

We selected two popular GitHub repositories (TensorFlow, Kubernetes) as experimental data. Using the GitHub API, we obtained issue information for specific time periods, including issue tags and all participating users, collecting 7,352 issues and 58,814 comment actions. Dataset details are shown in Table 2.

Table 2. Datasets of TensorFlow and Kubernetes

Repository	Issues	Comments	Developers	Label Attributes	Time Bucket
TensorFlow	2,459	13,067	-	-	3/22/2020-7/28/2020
Kubernetes	4,893	45,747	-	-	12/28/2020-7/30/2021

3.2 Evaluation Metrics

We evaluate algorithm performance using recall, precision, and F-Score. Recall is calculated as in Equation (9), precision as in Equation (10), and F-Score as in Equation (11), where Issue$_s$ is the test set, R_i is the Top-N recommendation list based on developer training behavior, and T_i is the actual set of users who participated in the issue.

3.3 Experimental Setup

Experiments were conducted using Python 3.8, PyTorch 1.9.0, CUDA 11.4, and RTX 3090. The dataset was split into training, validation, and test sets at a ratio of 8:1:1. GAT-UCG hyperparameters are shown in Table 3. Training ran for 2,000 epochs with early stopping if validation metrics didn' t improve for 100 epochs. Baseline methods GAT and GCN used identical embedding dimensions, learning rates, batch sizes, and attention heads, with other hyperparameters matching their original papers. The goal was to recommend top-N developers for each test issue, evaluated using Recall@N, Precision@N, and F-Score@N.

Table 3. Hyperparameter Settings

Hyperparameter	Value
Issue/Developer Embedding Dimension	237 (TensorFlow) / 363 (Kubernetes)
Hidden Layers	2
Attention Heads	4
Dropout Rate	0.2
L2 Regularization	0.0005

3.4 Ablation Study

To evaluate multi-head attention effectiveness, we compared single-head vs. multi-head models. Since 99.79% of issues have $ $10 participants, we limited recommendation lists to 10 and evaluated Top-N recall. Results in Table 4 show multi-head attention outperforms single-head by 8.57% average recall improvement across both repositories, demonstrating that multi-head attention stabilizes neighbor information propagation and enhances performance.

Table 4. Top-N Recommendation Recall with Different Attention Heads

Attention Head	TensorFlow	Kubernetes
Multi-head	39.67%, 48.29%, 52.85%, 55.59%, 58.89%	45.01%, 56.16%, 61.18%, 64.05%, 66.04%
Single-head	27.66%, 39.07%, 44.43%, 46.82%, 51.78%	34.55%, 46.35%, 53.29%, 57.29%, 60.77%

3.5 Experimental Results

Based on the ablation study, we applied multi-head attention to GAT-UCG and compared performance for Top-2, 4, 6, 8, 10 recommendations across methods. Results are shown in Tables 5-7.

Table 5. Top-N Recommendation Recall

Method	TensorFlow	Kubernetes
GAT-UCG	39.67%, 48.29%, 52.85%, 55.59%, 58.89%	45.01%, 56.16%, 61.18%, 64.05%, 66.04%
GAT [26]	16.14%, 28.32%, 36.80%, 43.65%, 51.27%	28.70%, 39.59%, 45.27%, 49.10%, 51.27%
GCN [22]	11.63%, 20.68%, 28.28%, 33.36%, 37.27%	12.18%, 22.17%, 29.11%, 35.21%, 37.27%
SN [1]	22.69%, 26.10%, 28.93%, 30.54%, 31.39%	18.04%, 21.08%, 23.41%, 25.58%, 26.22%
MF [17]	5.11%, 14.82%, 21.45%, 24.82%, 25.52%	4.22%, 8.31%, 12.08%, 14.98%, 17.13%

Table 6. Top-N Recommendation Precision

Method	TensorFlow	Kubernetes
GAT-UCG	39.80%, 24.22%, 17.62%, 13.94%, 11.42%	53.05%, 33.10%, 23.86%, 18.71%, 15.42%
GAT [26]	16.20%, 14.23%, 12.30%, 10.95%, 11.67%	33.30%, 22.97%, 17.51%, 14.24%, 11.90%
GCN [22]	10.37%, 9.45%, 8.36%, 7.57%, 8.93%	14.13%, 12.86%, 11.26%, 10.21%, 9.21%
SN [1]	12.84%, 12.56%, 10.78%, 9.90%, 8.61%	18.92%, 12.30%, 8.41%, 8.21%, 7.36%
MF [17]	7.31%, 13.59%, 14.34%, 13.13%, 11.67%	6.64%, 7.42%, 8.65%, 7.95%, 6.74%

Table 7. Top-N Recommendation F-Score

Method	TensorFlow	Kubernetes
GAT-UCG	39.73%, 32.26%, 26.43%, 22.30%, 19.02%	48.70%, 41.65%, 34.33%, 28.96%, 25.25%
GAT [26]	16.17%, 18.94%, 18.43%, 17.51%, 14.04%	30.82%, 29.25%, 25.25%, 22.08%, 19.32%
GCN [22]	11.64%, 13.81%, 14.16%, 13.36%, 12.26%	13.08%, 16.27%, 16.24%, 15.82%, 14.04%
SN [1]	23.96%, 18.28%, 14.75%, 12.13%, 10.22%	18.47%, 15.54%, 12.59%, 10.67%, 9.57%
MF [17]	5.62%, 8.26%, 9.59%, 9.92%, 8.03%	5.62%, 8.26%, 9.59%, 9.92%, 8.03%

3.6 Result Analysis

Visualized in Figures 6-8, results show GAT-UCG outperforms all four baselines on both repositories. In TensorFlow, recall improved by up to 39.61%; in Kubernetes, by up to 54.61%.

Key observations: 1. GNN-based models generally outperform matrix factorization and social network-based models, indicating GNNs effectively mine collaboration and preference features. 2. Among GNN models, integrating both developer collaboration and developer-issue participation information yields better performance than using only developer-issue participation. 3. GNN models perform best on Kubernetes, which has more users, interactions, and label features, suggesting richer graph content produces more informative embeddings and better results. Performance gains are more pronounced on larger datasets. 4. The social network-based user profile model performs better than GAT for Top-2 recommendations in TensorFlow due to more fixed participants and fewer label types. However, from Top-4 onward, GAT-UCG surpasses it, demonstrating GNNs' better adaptability to complex scenarios.

4 Conclusion

This paper proposes a graph attention network-based participant recommendation model for issue resolution, validated on popular GitHub repositories using precision, recall, and F-Score. Experiments show GAT-UCG outperforms GAT, GCN, SN, and MF baselines, effectively recommending participants by incorporating developer collaboration relationships.

Future work includes incorporating more issue features, as many issues lack complete tags. Enhancing feature representation could further improve recommendation performance.

References

[1] Liu Ye-hui, Zhao Hai-yan, Cao Jian, et al. Participants recommendation approaches for issue solving process in open source community [J]. Journal of Chinese Computer Systems, 2020, 41 (9): 1930-1934.

[2] Marlow J, Dabbish L, Herbsleb J. Impression formation in online peer production: Activity traces and personal profiles in github [C]// Proceedings of the 2013 Conference on Computer Supported Cooperative Work (CSCW' 13). New York, NY, USA, 2013: 117-128.

[3] Tsay J, Dabbish L, Herbsleb J. Influence of social and technical factors for evaluating contribution in github [C]// Proceedings of the 36th International Conference on Software Engineering (ICSE 2014). New York, NY, USA, 2014: 356-366.

[4] Yeh R T, Ramamoorthy C V. Proceedings of the 2nd international conference on software engineering [J]. Scientific American, 1976, 172 (4): 297-301.

[5] Montandon J E, Silva L L, Valente M T. Identifying experts in software libraries and frameworks among GitHub users [C]// Proceedings of the 16th International Conference on Mining Software Repositories, IEEE Press, 2019: 276-287.

[6] Zhou Tao, Wang Chao. A research on knowledge contribution behaviors of open source software community users [J]. Science Research Management, 2020, 41 (02): 202-209.

[7] Morris M R, Teevan J, Panovich K. A comparison of information seeking using search engines and social networks [C]// Fourth International AAAI Conference on Weblogs and Social Media. 2010.

[8] White R W, Richardson M, Liu Y. Effects of community size and contact rate in synchronous social Q&A [C]// Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. 2011: 2837-2846.

[9] Alami A, Dittrich Y, Wasowski A. Influencers of quality assurance in an open source community [C]// 2018 IEEE/ACM 11th International Workshop

on Cooperative and Human Aspects of Software Engineering (CHASE). IEEE, 2018: 61-68.

[10] Sowe S K, Stamelos I, Angelis L. Understanding knowledge sharing activities in free/open source software projects: An empirical study [J]. Journal of Systems and Software, 2008, 3 (81): 431-446.

[11] Wang H, Wang T, Yin G, et al. Linking issue tracker with Q&A sites for knowledge sharing across communities [J]. IEEE Transactions on Services Computing, 2015, 11 (5): 782-795.

[12] Guo L, Chen Q, Han W, et al. Collaborative topic prediction model for user interest recommendation in online social networks [J]. International Journal of Digital Content Technology & its Applications, 2012, 6 (23): 62-73.

[13] Wang Wei, Zhou Tianyi, Zhao Shengyu, Fan Jiakuan. Research on the development of global open source ecology [J]. Science Research Management, 2020, 41 (02): 202-209.

[14] Jiang J, Yang Y, He J, et al. Who should comment on this pull request? analyzing attributes for more accurate commenter recommendation in pull-based development [J]. Information and Software Technology, 2017, 84: 48-62.

[15] Chen T, Cai J, Wang H, et al. Instant expert hunting: building an answerer recommender system for a large scale Q&A website [C]// Proceedings of the 29th Annual ACM Symposium on Applied Computing. 2014: 260-265.

[16] Davoodi E, Afsharchi M, Kianmehr K. A social network-based approach to expert recommendation system [C]// International conference on hybrid artificial intelligence systems. Springer, Berlin, Heidelberg, 2012: 660-671.

[17] Xu C. A novel recommendation method based on social network using matrix factorization technique [J]. Information processing & management, 2018, 54 (3): 463-474.

[18] Cheng Long, Li Han. A review of recommendation algorithms based on matrix factorization [J]. Journal of Beijing Information Science & Technology University, 2021, 36 (02): 38-45+51.

[19] Qin Chuan, Zhu Hengshu, Zhuang Fuzhen, et al. A survey on knowledge graph-based recommender systems [J]. Scientia Sinica (Informationis), 2020, 50 (07): 937-956.

[20] Zhou Wanzhen, Cao Di, Xu Yunfeng, Liu Bin. A survey of recommendation systems [J]. Journal of Hebei University of Science and Technology, 2020, 41 (01): 76-87.

[21] Wu Z, Pan S, Chen F, et al. A comprehensive survey on graph neural networks [J]. IEEE transactions on neural networks and learning systems, 2020, 32 (1): 4-24.

[22] Kipf T N, Welling M. Semi-supervised classification with graph convolutional networks [J]. arXiv preprint arXiv: 1609. 02907, 2016.

[23] Zhang M, Chen Y. Link prediction based on graph neural networks [J]. Advances in Neural Information Processing Systems, 2018, 31: 5165-5175.

[24] Zhang S, Yin H, Chen T, et al. Gcn-based user representation learning for unifying robust recommendation and fraudster detection [C]// Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval. 2020: 689-698.

[25] Hamilton W L, Ying R, Leskovec J. Inductive representation learning on large graphs [C]// Proceedings of the 31st International Conference on Neural Information Processing Systems. 2017: 1025-1035.

[26] Veličković P, Cucurull G, Casanova A, et al. Graph attention networks [J]. arXiv preprint arXiv: 1710. 10903, 2017.

[27] Tao Z, Wei Y, Wang X, et al. MGAT: multimodal graph attention network for recommendation [J]. Information Processing & Management, 2020, 57 (5): 102277.

[28] Fan W, Ma Y, Li Q, et al. Graph neural networks for social recommendation [C]// The World Wide Web Conference. 2019: 417-426.

[29] Guo Z, Wang H. A deep graph neural network-based mechanism for social recommendations [J]. IEEE Transactions on Industrial Informatics, 2020, 17 (4): 2776-2783.

[30] Wang J, Xie H, Wang F L, et al. Top-N personalized recommendation with graph neural networks in MOOCs [J]. Computers and Education: Artificial Intelligence, 2021, 2: 10001.

Note: Figure translations are in progress. See original paper for figures.

Source: ChinaXiv —Machine translation. Verify with original.