Collaboration-Aware Graph Convolutional Network for Recommender Systems

Yu Wang
yu.wang.1@vanderbilt.edu
Vanderbilt University

Yuying Zhao
yuying.zhao@vanderbilt.edu
Vanderbilt University

Yi Zhang
yi.zhang@vanderbilt.edu
Vanderbilt University

Tyler Derr
derr.tyler@vanderbilt.edu
Vanderbilt University

ABSTRACT

Graph Neural Networks (GNNs) have been successfully adopted in recommender systems by virtue of the message-passing that implicitly captures collaborative effect. Nevertheless, most of the existing message-passing mechanisms for recommendation are directly inherited from GNNs without scrutinizing whether the captured collaborative effect would benefit the prediction of user preferences. In this paper, we first analyze how message-passing captures the collaborative effect and propose a recommendation-oriented topological metric, Common Interacted Ratio (CIR), which measures the level of interaction between a specific neighbor of a node with the rest of its neighbors. After demonstrating the benefits of leveraging collaborations from neighbors with higher CIR, we propose a recommendation-tailored GNN, Collaboration-Aware Graph Convolutional Network (CAGCN), that goes beyond 1-Weisfeiler-Lehman(1-WL) test in distinguishing non-bipartite-subgraph-isomorphic graphs. Experiments on six benchmark datasets show that the best CAGCN variant outperforms the most representative GNN-based recommendation model, LightGCN, by nearly 10% in Recall@20 and also achieves around 80% speedup. Our code/supplementary is at https://github.com/YuWVandy/CAGCN.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning**.

KEYWORDS

Recommender systems, graph neural networks, collaborative effect

ACM Reference Format:
Yu Wang, Yuying Zhao, Yi Zhang, and Tyler Derr. 2023. Collaboration-Aware Graph Convolutional Network for Recommender Systems. In *Proceedings of the ACM Web Conference 2023 (WWW '23), May 1–5, 2023, Austin, TX, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3543507.3583229

1 INTRODUCTION

Recommender systems aim to alleviate information overload by helping users discover items of interest [2, 47] and have been widely deployed in real-world applications [31]. Given historical user-item interactions (e.g., click, purchase, review, and rate), the key is to leverage the collaborative effect [3, 11, 37] to predict how likely users will interact with items. A common paradigm for modeling collaborative effect is to first learn embeddings of users/items capable of recovering historical user-item interactions and then perform top-K recommendation based on the pairwise similarity between the learned user/item embeddings.

Since historical user-item interactions can be naturally represented as a bipartite graph with users/items being nodes and interactions being edges [9, 16, 37] and given the unprecedented success of GNNs in learning node representations [12, 13, 18, 53], recent research has started to leverage GNNs to learn user/item embeddings for the recommendation. Two pioneering works NGCF [37] and LightGCN [9] leverage graph convolutions to aggregate messages from local neighborhoods, which directly injects the collaborative signal into user/item embeddings. More recently, [1, 44] explore the robustness and self-supervised learning [40] of graph convolution for recommendation. However, the message-passing mechanisms in all previous recommendation models are directly inherited from GNNs without carefully justifying how collaborative signals are captured and whether the captured collaborative signals would benefit the prediction of user preference. Such ambiguous understanding on how the message-passing captures collaborative signals would pose the risk of learning uninformative or even harmful user/item representations when adopting GNNs in recommendation. For example, [4] shows that a large portion of user interactions cannot reflect their actual purchasing behaviors. In this case, blindly passing messages following existing styles of GNNs could capture harmful collaborative signals from these unreliable interactions, which hinders the performance of GNN-based recommender systems.

To avoid collecting noisy or even harmful collaborative signals in message-passing of traditional GNNs, existing work GTN [4] proposes to adaptively propagate user/item embeddings by adjusting the weight of edges based on items' similarity to users' main preferences (i.e., the trend). However, such similarity is computed based on the learned embeddings that still implicitly encode noisy collaborative signals from unreliable user-item interactions. Worse still, calculating edge weights based on user/item embeddings along the training on the fly is computationally prohibitive and hence prevents the model from being deployed in industrial-level recommendations. SGCN [1] attaches the message-passing with a trainable stochastic binary mask to prune noisy edges. However, the unbiased gradient estimator increases the computational load.

WWW '23, May 1–5, 2023, Austin, TX, USA

ACM ISBN 978-1-4503-9416-1/23/04…$15.00
https://doi.org/10.1145/3543507.3583229

Despite the fundamental importance of capturing beneficial collaborative signals, the related studies are still in their infancy. To fill this crucial gap, we aim to demystify the collaborative effect captured by message-passing and develop new insights towards customizing message-passing for recommendations. Furthermore, these insights motivate us to design a recommendation-tailored GNN, Collaboration-Aware Graph Convolutional Network(CAGCN), that passes neighborhood information based on their Common Interacted Ratio (CIR) via the Collaboration-Aware Graph Convolution (CAGC). Our major contributions are listed as follows:

- **Novel Perspective on Collaborative Effect:** We demystify the collaborative effect by analyzing how message-passing helps capture collaborative signals and when the captured collaborative signals are beneficial in computing users' ranking over items.
- **Novel Recommendation-tailored Topological Metric:** We then propose a recommendation-tailored topological metric, Common Interacted Ratio (CIR), and demonstrate the capability of CIR to quantify the benefits of the messages from neighborhoods.
- **Novel Convolution beyond 1-WL for Recommendation:** We integrate CIR into message-passing and propose a novel Collaboration-Aware Graph Convolutional Network (CAGCN). Then we prove that it can go beyond 1-WL test in distinguishing non-bipartite-subgraph-isomorphic graphs, show its superiority on real-world datasets including two newly collected ones, and provide an in-depth interpretation of its advantages.

Next, we comprehensively analyze the collaborative effect captured by message-passing and propose CIR to measure whether the captured collaborative effect benefits the prediction of user preferences.

2 ANALYSIS ON COLLABORATIVE EFFECT

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be the user-item bipartite graph, where the node set $\mathcal{V} = \mathcal{U} \cup \mathcal{I}$ includes the user set $\mathcal{U}$ and the item set $\mathcal{I}$. Following previous work [9, 20, 37], we only consider the implicit user-item interactions and denote them as edges $\mathcal{E}$ where e_{pq} represents the edge between node p and q. The network topology is described by its adjacency matrix $\mathbf{A} \in \{0,1\}^{|\mathcal{V}|\times|\mathcal{V}|}$, where $\mathbf{A}_{pq} = 1$ when $e_{pq} \in \mathcal{E}$, and $\mathbf{A}_{pq} = 0$ otherwise. Let $\mathcal{N}_p^l$ denote the set of observed neighbors that are exactly l-hops away from p and $\mathcal{S}_p = (\mathcal{V}_{\mathcal{S}_p}, \mathcal{E}_{\mathcal{S}_p})$ be the neighborhood subgraph [43] induced in $\mathcal{G}$ by $\widetilde{\mathcal{N}}_p^1 = \mathcal{N}_p^1 \cup \{p\}$. We use $\mathscr{P}_{pq}^l$ to denote the set of shortest paths of length l between node p and q and denote one of such paths as P_{pq}^l. Note that $\mathscr{P}_{pq}^l = \emptyset$ if it is impossible to have a path between p and q of length l, e.g., $\mathscr{P}_{11}^1 = \emptyset$ in an acyclic graph. Furthermore, we denote the initial embeddings of users/items as $\mathbf{E}^0 \in \mathbb{R}^{(n+m)\times d^0}$ where $\mathbf{e}_p^0 = \mathbf{E}_p^0$ and d_p are the node p's embedding and degree.

Following [9, 37], each node has no semantic features but purely learnable embeddings. Therefore, we remove the nonlinear transformation by leveraging LightGCN [9] as the canonical architecture and exclusively explore the collaborative effect captured by message-passing. LightGCN passes messages from user u/item i's neighbors within L-hops to u/i:

$$\mathbf{e}_u^{l+1} = d_u^{-0.5} \sum_{j\in\mathcal{N}_u^1} d_j^{-0.5}\mathbf{e}_j^l, \mathbf{e}_i^{l+1} = d_i^{-0.5} \sum_{v\in\mathcal{N}_i^1} d_v^{-0.5}\mathbf{e}_v^l, \tag{1}$$

$\forall l \in \{0, ..., L\}$. The propagated embeddings at all layers including the original embedding are aggregated together via mean-pooling:

$$\mathbf{e}_u = \frac{1}{(L+1)}\sum_{l=0}^{L}\mathbf{e}_u^l,\ \mathbf{e}_i = \frac{1}{(L+1)}\sum_{l=0}^{L}\mathbf{e}_i^l, \forall u \in \mathcal{U}, \forall i \in \mathcal{I} \tag{2}$$

In the training stage, for each observed user-item interaction (u, i), LightGCN randomly samples a negative item i^- that u has never interacted with before, and forms the triple (u, i, i^-), which collectively forms the set of observed training triples O. After that, the ranking scores of the user over these two items are computed as $y_{ui} = \mathbf{e}_u^\top \mathbf{e}_i$ and $y_{ui^-} = \mathbf{e}_u^\top \mathbf{e}_{i^-}$, which are finally used in optimizing the pairwise Bayesian Personalized Ranking (BPR) loss [27]:

$$\mathcal{L}_{\text{BPR}} = \sum_{(u,i,i^-)\in O} -\ln\sigma(y_{ui} - y_{ui^-}), \tag{3}$$

where $\sigma(\cdot)$ is the Sigmoid function, and we omit the L_2 regularization here since it is mainly for alleviating overfitting and has no influence on the collaborative effect captured by message passing.

Under the above LightGCN framework, we expect to answer the following two questions:

- $\boldsymbol{Q_1}$: How does message-passing capture the collaborative effect and leverage it in computing users' ranking?
- $\boldsymbol{Q_2}$: When do collaborations captured by message-passing benefit the computation of users' ranking over items?

Next, We address $\boldsymbol{Q_1}$ by theoretically deriving users' ranking over items under the message-passing framework of LightGCN and address $\boldsymbol{Q_2}$ by proposing the Common Interacted Ratio (CIR) to measure the benefits of leveraging collaborations from each neighbor in computing users' ranking. The answers to the above two questions further motivate our design of Collaboration-Aware Graph Convolutional Network in Section 3.

2.1 How does message-passing capture collaborative effect?

The collaborative effect occurs when the prediction of a user's preference relies on other users' preferences or items' properties [28]. Therefore, to answer $\boldsymbol{Q_1}$, we need to seek whether we leverage other nodes' embeddings in computing a specific user's ranking over items. In the inference stage of LightGCN, we take the inner product between user u's embedding and item i's embedding after L-layers' message-passing to compute the ranking as[1]:

$$y_{ui}^L = (\sum_{l_1=0}^{L}\sum_{j\in\mathcal{N}_u^{l_1}}\sum_{l_2=l_1}^{L}\beta_{l_2}\alpha_{ju}^{l_2}\mathbf{e}_j^0)^\top(\sum_{l_1=0}^{L}\sum_{v\in\mathcal{N}_i^{l_1}}\sum_{l_2=l_1}^{L}\beta_{l_2}\alpha_{vi}^{l_2}\mathbf{e}_v^0), \tag{4}$$

where $\alpha_{ju}^{l_2} = \sum_{P_{ju}^{l_2}\in\mathscr{P}_{ju}^{l_2}}\prod_{e_{pq}\in P_{ju}^{l_2}} d_p^{-0.5}d_q^{-0.5}$ ($\alpha_{ju}^{l_2} = 0$ if $\mathscr{P}_{ju}^{l_2} = \emptyset$) denotes the total weight of all paths of length l_2 from j to u, $\mathcal{N}_u^0 = \{u\}$ and specifically, $\alpha_{uu}^0 = 1$. β_{l_2} is the weight measuring contributions of propagated embeddings at layer l_2. Thus, based on Eq. (4), we present the answer to $\boldsymbol{Q_1}$ as $\boldsymbol{A_1}$: *L-layer LightGCN-based message-passing captures collaborations between pairs of nodes $\{(j,v)|j \in \bigcup_{l=0}^{L}\mathcal{N}_u^l, v \in \bigcup_{l=0}^{L}\mathcal{N}_i^l\}$, and the collaborative strength of each pair is determined by 1) $\mathbf{e}_j^{0\top}\mathbf{e}_v^0$: embedding similarity between j and v, 2) $\{\alpha_{ju}^l\}_{l=0}^L(\{\alpha_{vi}^l\}_{l=0}^L)$: weight of all paths of length l to L from j to u (v to i), and 3) $\{\beta_l\}_{l=0}^L$: the weight of each layer.*

[1] Detailed derivation is attached in Appendix A.2.

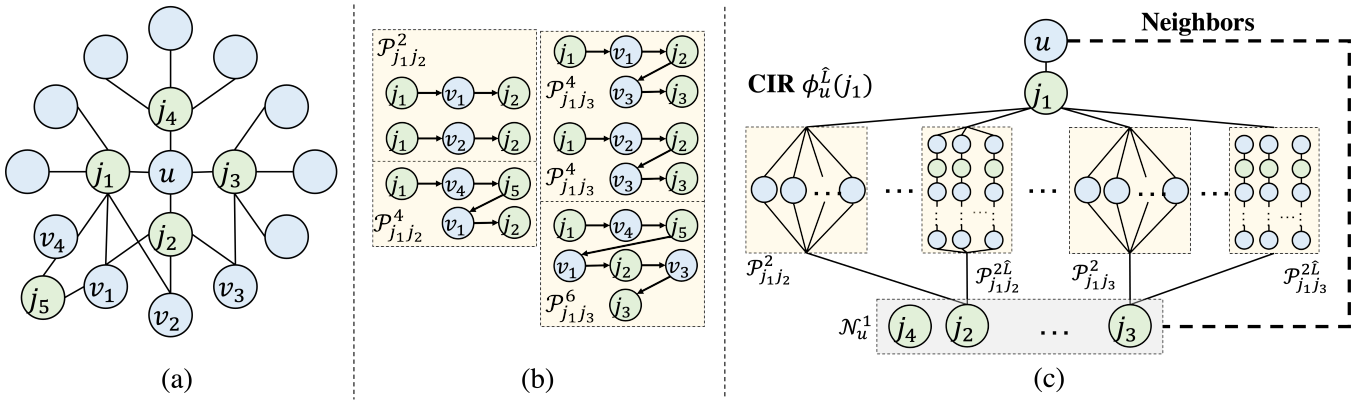


Figure 1: In (a)-(b), j_1 has more interactions (paths) with (to) u's neighborhood than j_4 and hence is more representative of u's purchasing behaviors than j_4. In (c), we quantify CIR between j_1 and u via the paths (and associated nodes) between j_1 and $\mathcal{N}_u^1$.

2.2 When is the captured collaborative effect beneficial to users' ranking?

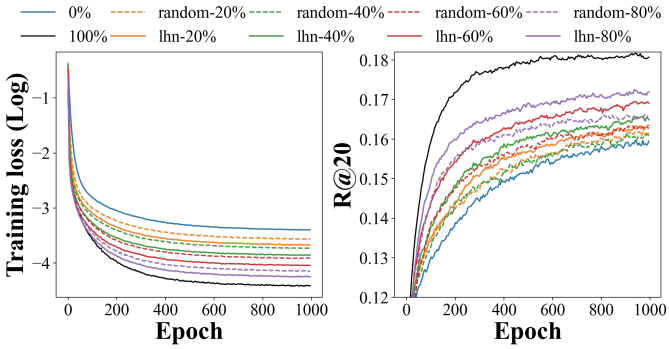
Although users could leverage collaborations from other users/items as demonstrated above, we cannot guarantee all of these collaborations benefit the prediction of their preferences. For example, in Figure 1(a)-(b), u's interacted item j_1 has more interactions (paths) to u's neighborhoods than j_4 and hence is more representative of u's purchasing behaviors [1, 4]. For each user u, we propose the Common Interacted Ratio to quantify the level of interaction between each specific neighbor of u and u's whole item neighborhood:

Definition 1. Common Interacted Ratio (CIR): For any item $j \in \mathcal{N}_u^1$ of user u, the CIR of j around u considering nodes up to $(\widehat{L}+1)$-hops away from u, i.e., $\phi_u^{\widehat{L}}(j)$, is defined as the average interacted ratio of j with all neighboring items of u in $\mathcal{N}_u^1$ through paths of length $\leq 2\widehat{L}$:

$$\phi_u^{\widehat{L}}(j) = \frac{1}{|\mathcal{N}_u^1|} \sum_{i \in \mathcal{N}_u^1} \sum_{l=1}^{\widehat{L}} \alpha^{2l} \sum_{P_{ji}^{2l} \in \mathscr{P}_{ji}^{2l}} \frac{1}{f(\{\mathcal{N}_k^1 | k \in P_{ji}^{2l}\})}, \tag{5}$$

$\forall j \in \mathcal{N}_u^1, \forall u \in \mathcal{U}$, where $\{\mathcal{N}_k^1 | k \in P_{ji}^{2l}\}$ represents the set of the 1-hop neighborhood of node k along the path P_{ji}^{2l} from node j to i of length $2l$ including i, j. f is a normalization function to differentiate the importance of different paths in $\mathscr{P}_{ji}^{2l}$ and its value depends on the neighborhood of each node along the path P_{ji}^{2l}. α^{2l} is the importance of paths of length $2l$.

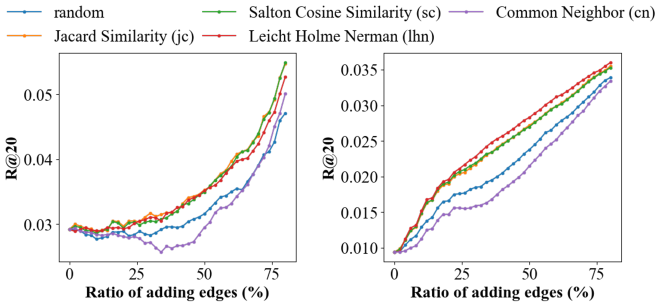
As shown in Figure 1(c), $\phi_u^{\widehat{L}}(j_1)$ is decided by paths of length between 2 to $2\widehat{L}$. By configuring different $\widehat{L}$ and f, $\sum_{P_{ji}^{2l} \in \mathscr{P}_{ji}^{2l}} \frac{1}{f(\{\mathcal{N}_k^1 | k \in P_{ji}^{2l}\})}$ could express many graph similarity metrics [15, 17, 21, 30, 52] and we discuss them in Appendix A.1. For simplicity, henceforth we denote $\phi_u^{\widehat{L}}(j)$ as $\phi_u(j)$. We next empirically verify the importance of leveraging collaborations from neighbors with higher CIR by incrementally adding edges into an initially edge-less graph according to their CIR and visualizing the performance change. Specifically, we consider the performance change in two settings, retraining and pretraining, which are visualized in Figure 2 and 3, respectively. In both of these two settings, we iteratively cycle each node and add its corresponding neighbor according to the CIR until hitting the budget. Here we consider variants of CIR that we later define in Section 4.1 with further details in Appendix A.1.



(a) Gowalla-Loss (b) Gowalla-Recall

Figure 2: The training loss (left) is lower and the performance (right) is higher when adding edges according to the variant CIR-lhn (Leicht Holme Nerman) than adding randomly under the same addition budget. Detailed experimental settings and more results are provided in Appendix A.5.1.

For the re-training setting, we first remove all observed edges in the training set to create the edgeless bipartite graph and then incrementally add edges according to their CIR and retrain user/item embeddings. In Figure 2, we evaluate the performance on the newly constructed bipartite graph under different edge budgets. Clearly, the training loss/performance becomes lower/higher when adding more edges because message-passing captures more collaborative effects. Furthermore, since edges with higher CIR connect neighbors with more connections to the whole neighborhood, optimizing embeddings of nodes incident to these edges pull the whole neighborhood closer and hence leads to the lower training loss over neighborhoods' connections, which causes the overall lower training loss in Figure 2(a). In Figure 2(b), we observe that under the same adding budget, adding according to CIRs achieves higher performance than adding randomly. It is because neighbors with higher interactions with the whole neighborhood are more likely to have higher interactions with neighbors to be predicted (We empirically verify this in Table 5.). Then for each user, maximizing its embedding similarity to its training neighbors with higher CIR will indirectly improve its similarity to its to-be-predicted neighbors, which leads to lower population risk and higher generalization/performance.



(a) Loseit-Recall (b) Amazon-Recall

Figure 3: The performance of adding edges according to CIR variants generally increases faster than adding randomly after pre-training. See Appendix A.5.2 for more results.

For the pre-training setting, we first pre-train user/item embeddings on the original bipartite graph and then propagate the pre-trained embeddings on the newly constructed bipartite graph under different edge budgets. This setting is more realistic since in the real world, with the exponential interactions streamingly coming in [36] while the storage space is limited, we are forced to keep only partial interactions and the pre-trained user/item embeddings. Figure 3 demonstrates that under the same adding budget, keeping edges according to CIR leads to higher performance than keeping randomly, which further verifies the effectiveness of CIR in quantifying the edge importance. An interesting observation is that adding more edges cannot always bring performance gain as shown in Figure 3(a) when the ratio of added edges is between 0%-20%. We hypothesize there are two reasons. From network topology, only when edges are beyond a certain level can the network form a giant component so that users could receive enough neighborhood information. Secondly, from representation learning, more nodes would have inconsistent neighborhood contexts between the training and the inference when only a few edges are added. Such inconsistent neighborhood context would compromise the performance and will be alleviated when more edges are added as shown later in Figure 3(a). Furthermore, different CIR variants cause different increasing speeds of performance. For example, sc is faster on Loseit in Figure 3(a) while lhn is faster on Amazon in Figure 3(b). Except for the cn, jc/sc/lhn lead to faster improvement than the random one, which highlights the potential of CIR in devising cost-effective strategies for pruning edges in the continual learning [35].

From the above analysis, we summarize the answer $\boldsymbol{A_2}$ to $\boldsymbol{Q_2}$ as: *Leveraging collaborations from u's neighboring node j with higher CIR $\phi_u(j)$ would cause more benefits to u's ranking.*

3 COLLABORATION-AWARE GRAPH CONVOLUTIONAL NETWORKS

The former section demonstrates that passing messages according to neighbors' CIR is crucial in improving users' ranking. This motivates us to propose a new graph convolution operation, Collaboration-Aware Graph Convolution(CAGC), which passes node messages based on the benefits of their provided collaborations. Furthermore, we wrap the proposed CAGC within LightGCN and develop the Collaboration-Aware Graph Convolutional Network (CAGCN).

3.1 Collaboration-Aware Graph Convolution

The core idea of CAGC is to strengthen/weaken the messages passed from neighbors with higher/lower CIR to center nodes. To achieve this, we compute the edge weight as:

$$\mathbf{\Phi}_{ij} = \begin{cases} \phi_i(j), & \text{if } \mathbf{A}_{ij} > 0 \\ 0, & \text{if } \mathbf{A}_{ij} = 0 \end{cases}, \forall i, j \in \mathcal{V} \tag{6}$$

where $\phi_i(j)$ is the CIR of neighboring node j centering around i. Note that unlike the symmetric graph convolution $\mathbf{D}^{-0.5}\mathbf{A}\mathbf{D}^{-0.5}$ used in LightGCN, here $\mathbf{\Phi}$ is unsymmetric. This is rather interpretable: the interacting level of node j with i's neighborhood is likely to be different from the interacting level of node i with j's neighborhood. We further normalize $\mathbf{\Phi}$ and combine it with the LightGCN convolution:

$$\mathbf{e}_i^{l+1} = \sum_{j \in \mathcal{N}_i^1} g(\gamma_i \frac{\mathbf{\Phi}_{ij}}{\sum_{k \in \mathcal{N}_i^1} \mathbf{\Phi}_{ik}}, d_i^{-0.5} d_j^{-0.5}) \mathbf{e}_j^l, \forall i \in \mathcal{V} \tag{7}$$

where γ_i is a coefficient that varies the total amount of messages flowing to node i and controls its embedding magnitude [24]. g is a function combining the edge weights computed based on CIR and LightGCN. We could either simply set g as the weighted summation of these two propagated embeddings or learn g by parametrization. Next, we prove that for certain choices of g, CAGCN can go beyond 1-WL in distinguishing non-bipartite-subgraph-isomorphic graphs. First, we prove the equivalence between the subtree-isomorphism and the subgraph-isomorphism in bipartite graphs:

Theorem 1. *In bipartite graphs, two subgraphs that are subtree-isomorphic if and only if they are subgraph-isomorphic*[2].

Proof. We prove this theorem in two directions. Firstly ($\Longrightarrow$), we prove that in a bipartite graph, two subgraphs that are subtree-isomorphic are also subgraph-isomorphic by contradiction. Assuming that there exists two subgraphs $\mathcal{S}_u$ and $\mathcal{S}_i$ that are subtree-isomorphic yet not subgraph-isomorphic in a bipartite graph, i.e., $\mathcal{S}_u \cong_{subtree} \mathcal{S}_i$ and $\mathcal{S}_u \ncong_{subgraph} \mathcal{S}_i$. By definition of subtree-isomorphism, we trivially have $\mathbf{e}_v^l = \mathbf{e}_{h(v)}^l, \forall v \in \mathcal{V}_{\mathcal{S}_u}$. Then to guarantee $\mathcal{S}_u \ncong_{subgraph} \mathcal{S}_i$ and also since edges are only allowed to connect u and its neighbors $\mathcal{N}_u^1$ in the bipartite graph, there must exist at least an edge e_{uv} between u and one of its neighbors $v \in \mathcal{N}_u^1$ such that $e_{uv} \in \mathcal{E}_{\mathcal{S}_u}, e_{h(u)h(v)} \notin \mathcal{E}_{\mathcal{S}_i}$, which contradicts the assumption that $\mathcal{S}_\sqcap \cong_{subtree} \mathcal{S}_\rangle$. Secondly ($\Longleftarrow$), we can prove that in a bipartite graph, two subgraphs that are subgraph-isomorphic are also subtree-isomorphic, which trivially holds since in any graph, subgraph-isomorphism leads to subtree-isomorphism [43]. □

Since 1-WL test can distinguish subtree-isomorphic graphs [43], the equivalence between these two isomorphisms indicates that in bipartite graphs, both of the subtree-isomorphic graphs and subgraph-isomorphic graphs can be distinguished by 1-WL test. Therefore, to go beyond 1-WL in bipartite graphs, we need to propose a novel graph isomorphism, bipartite-subgraph-isomorphism in Definition 2, which is even harder to be distinguished than the subgraph-isomorphism by 1-WL test.

[2]Definitions of subtree-/subgraph-isomorphism are in Supplementary B.4[43].

Definition 2. Bipartite-subgraph-isomorphism: $\mathcal{S}_u$ and $\mathcal{S}_i$ are bipartite-subgraph-isomorphic, denoted as $\mathcal{S}_u \cong_{bi-subgraph} \mathcal{S}_i$, if there exists a bijective mapping $h : \widetilde{\mathcal{N}}_u^1 \cup \mathcal{N}_u^2 \rightarrow \widetilde{\mathcal{N}}_i^1 \cup \mathcal{N}_i^2$ such that $h(u) = i$ and $\forall v, v' \in \widetilde{\mathcal{N}}_u^1 \cup \mathcal{N}_u^2$, $e_{vv'} \in \mathcal{E} \iff e_{h(v)h(v')} \in \mathcal{E}$ and $\mathbf{e}_v^l = \mathbf{e}_{h(v)}^l, \mathbf{e}_{v'}^l = \mathbf{e}_{h(v')}^l$.

Lemma 1. *If g is multilayer perceptron (MLP), then we have that $g(\{(\gamma_i \widetilde{\mathbf{\Phi}}_{ij}, \mathbf{e}_j^l) | j \in \mathcal{N}_i^1\}, \{(d_i^{-0.5} d_j^{-0.5}, \mathbf{e}_j^l) | j \in \mathcal{N}_i^1\})$ is injective.*

Proof. If we assume that all node embeddings share the same discretization precision, then embeddings of all nodes in a graph can form a countable set $\mathcal{H}$. Similarly, for each edge in a graph, its CIR-based weight $\widetilde{\mathbf{\Phi}}_{ij}$ and degree-based weight $d_i^{-0.5} d_j^{-0.5}$ can also form two different countable sets $\mathcal{W}_1, \mathcal{W}_2$ with $|\mathcal{W}_1| = |\mathcal{W}_2|$. Then $\mathcal{P}_1 = \{\widetilde{\mathbf{\Phi}}_{ij} \mathbf{e}_i | \widetilde{\mathbf{\Phi}}_{ij} \in \mathcal{W}_1, \mathbf{e}_i \in \mathcal{H}\}, \mathcal{P}_2 = \{d_i^{-0.5} d_j^{-0.5} \mathbf{e}_i | d_i^{-0.5} d_j^{-0.5} \in \mathcal{W}_2, \mathbf{e}_i \in \mathcal{H}\}$ are also two countable sets. Let P_1, P_2 be two multisets containing elements from $\mathcal{P}_1$ and $\mathcal{P}_2$, respectively, and $|P_1| = |P_2|$. Then by Lemma 1 in [43], there exists a function s such that $\pi(P_1, P_2) = \sum_{p_1 \in P_1, p_2 \in P_2} s(p_1, p_2)$ is unique for any distinct pair of multisets (P_1, P_2). Since the MLP-based g is a universal approximator [45] and hence can learn s, we know that g is injective. □

Theorem 2. *Let M be a GNN with sufficient number of CAGC-based convolution layers defined by Eq. (7). If g is MLP, then M is strictly more expressive than 1-WL in distinguishing subtree-isomorphic yet non-bipartite-subgraph-isomorphic graphs.*

Proof. We prove this theorem in two directions. Firstly ($\Longrightarrow$), following [43], we prove that the designed CAGCN here can distinguish any two graphs that are distinguishable by 1-WL by contradiction. Assume that there exist two graphs $\mathcal{G}_1$ and $\mathcal{G}_2$ which can be distinguished by 1-WL but cannot be distinguished by CAGCN. Further, suppose that 1-WL cannot distinguish these two graphs in the iterations from 0 to $L-1$, but can distinguish them in the L^{th} iteration. Then, there must exist two neighborhood subgraphs $\mathcal{S}_u$ and $\mathcal{S}_i$ whose neighboring nodes correspond to two different sets of node labels at the L^{th} iteration, i.e., $\{\mathbf{e}_v^l | v \in \mathcal{N}_u^1\} \neq \{\mathbf{e}_j^l | j \in \mathcal{N}_i^1\}$. Since g is injective by Lemma 1, for $\mathcal{S}_u$ and $\mathcal{S}_i$, g would yield two different feature vectors at the L^{th} iteration. This means that CAGCN can also distinguish $\mathcal{G}_1$ and $\mathcal{G}_2$, which contradicts the assumption.

Secondly ($\Longleftarrow$), we prove that there exist at least two graphs that can be distinguished by CAGCN but cannot be distinguished by 1-WL. Figure 11 in Supplementary B.4 presents two of such graphs $\mathcal{S}_u, \mathcal{S}_u'$, which are subgraph isomorphic but non-bipartite-subgraph-isomorphic. Assuming u and u' have exactly the same neighborhood feature vectors $\mathbf{e}$, then directly propagating according to 1-WL or even considering node degree as the edge weight as GCN [12] can still end up with the same propagated feature for u and u'. However, if we leverage JC to calculate CIR as introduced in Appendix A.1, then we end up with $\{(d_u d_{j_1})^{-0.5}\mathbf{e}, (d_u d_{j_2})^{-0.5}\mathbf{e}, (d_u d_{j_3})^{-0.5}\mathbf{e}\} \neq \{(d_{u'}^{-0.5} d_{j_1'}^{-0.5} + \widetilde{\mathbf{\Phi}}_{u' j_1'})\mathbf{e}, (d_{u'}^{-0.5} d_{j_2'}^{-0.5} + \widetilde{\mathbf{\Phi}}_{u' j_2'})\mathbf{e}, (d_{u'}^{-0.5} d_{j_3'}^{-0.5} + \widetilde{\mathbf{\Phi}}_{u' j_3'})\mathbf{e}\}$. Since g is injective by Lemma 1, CAGCN would yield two different embeddings for u and u'. □

Theorem 2 indicates that GNNs whose aggregation scheme is CAGC can distinguish non-bipartite-subgraph-isomorphic graphs that are indistinguishable by 1-WL.

3.2 Model Architecture and complexity analysis

Following the principle of LightGCN that the designed graph convolution should be light and easy to train, except for the message-passing component, all other components of our proposed CAGCN is exactly the same as LightGCN including the average pooling and the model training, which have already been covered in Section 2. We provide the detailed time/space complexity comparison between our CAGCN and all other baselines in Appendix A.3.

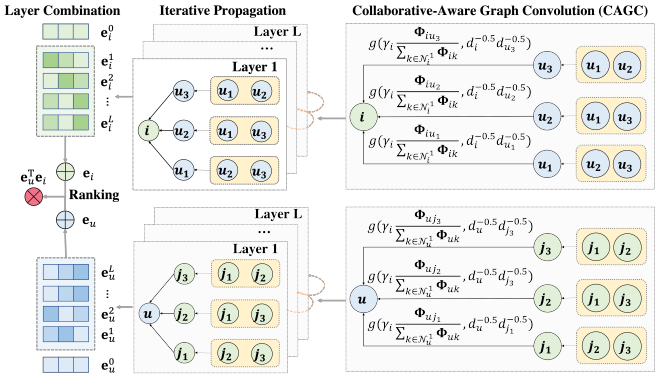


Figure 4: The architecture of the proposed CAGCN.

4 EXPERIMENTS

In this section, we conduct experiments to evaluate the effectiveness of our proposed CAGCN.

4.1 Experimental Settings

4.1.1 Datasets. Following [9, 37], we validate the proposed approach on four widely used benchmark datasets in recommender systems, including **Gowalla**, **Yelp**, **Amazon**, and **Ml-1M**, the details of which are provided in [9, 37]. Moreover, we collect two extra datasets to further demonstrate the superiority of our proposed model in even broader user-item interaction domains: **(1) Loseit**: This dataset is collected from subreddit *loseit - Lose the Fat*[3] from March 2020 to March 2022 where users discuss healthy and sustainable methods of losing weight via posts. To ensure the quality of this dataset, we use the 10-core setting [8], i.e., retaining users and posts with at least ten interactions. **(2) News**: This dataset includes the interactions from subreddit *World News*[4] where users share major news around the world via posts. Similarly, we use the 10-core setting to ensure the quality of this dataset. We summarize the statistics of all six datasets in Table 1.

Table 1: Basic dataset statistics.

Dataset	# Users	# Items	# Interactions	Density
Gowalla	29, 858	40, 981	1, 027, 370	0.084%
Yelp	31, 668	38, 048	1, 561, 406	0.130%
Amazon	52, 643	91, 599	2, 984, 108	0.062%
Ml-1M	6, 022	3, 043	895, 699	4.888%
Loseit	5, 334	54, 595	230, 866	0.08%
News	29, 785	21, 549	766, 874	0.119%

*__Yelp__: Yelp2018; *__Amazon__: Amazon-Books;*__Ml-1M__: Movielens-1M.

[3] https://www.reddit.com/r/loseit/
[4] https://www.reddit.com/r/worldnews/

Table 2: Performance comparison of CAGCN(*) with baselines. The best and runner-up results are in bold and underlined.

Model	Metric	MF	NGCF	LightGCN	UltraGCN	CAGCN				CAGCN*		
						-jc	-sc	-cn	-lhn	-jc	-sc	-lhn
Gowalla	Recall@20	0.1554	0.1563	0.1817	0.1867	0.1825	0.1826	0.1632	0.1821	**0.1878**	**0.1878**	0.1857
	NDCG@20	0.1301	0.1300	0.1570	0.1580	0.1575	0.1577	0.1381	0.1577	**0.1591**	0.1588	0.1563
Yelp2018	Recall@20	0.0539	0.0596	0.0659	0.0675	0.0674	0.0671	0.0661	0.0661	0.0708	**0.0711**	0.0676
	NDCG@20	0.0460	0.0489	0.0554	0.0553	0.0564	0.0560	0.0546	0.0555	0.0586	**0.0590**	0.0554
Amazon	Recall@20	0.0337	0.0336	0.0420	**0.0682**	0.0435	0.0435	0.0403	0.0422	0.0510	0.0506	0.0457
	NDCG@20	0.0265	0.0262	0.0331	**0.0553**	0.0343	0.0342	0.0321	0.0333	0.0403	0.0400	0.0361
Ml-1M	Recall@20	0.2604	0.2619	0.2752	0.2783	0.2780	0.2786	0.2730	0.2760	0.2822	**0.2827**	0.2799
	NDCG@20	0.2697	0.2729	0.2820	0.2638	0.2871	**0.2881**	0.2818	0.2871	0.2775	0.2776	0.2745
Loseit	Recall@20	0.0539	0.0574	0.0588	0.0621	0.0622	0.0625	0.0502	0.0592	0.0654	**0.0658**	**0.0658**
	NDCG@20	0.0420	0.0442	0.0465	0.0446	0.0474	0.0470	0.0379	0.0461	0.0486	0.0484	**0.0489**
News	Recall@20	0.1942	0.1994	0.2035	0.2034	0.2135	0.2132	0.1726	0.2084	**0.2182**	0.2172	0.2053
	NDCG@20	0.1235	0.1291	0.1311	0.1301	0.1385	0.1384	0.1064	0.1327	0.1405	**0.1414**	0.1311
Avg. Rank	Recall@20	9.83	9.17	7.33	4.17	4.67	4.33	8.83	6.17	1.67	**1.50**	3.33
	NDCG@20	9.50	9.17	5.83	6.00	3.67	4.00	8.33	5.00	**2.50**	**2.50**	5.17

jc-Jacard Similarity, **sc**-Salton Cosine Similarity, **cn**-Common Neighbors, **lhn**-Leicht-Holme-Nerman

4.1.2 Baseline methods. We compare our model with MF, NGCF, LightGCN, UltraGCN, GTN [4, 9, 20, 27, 37]. Details of them are clarified in Appendix A.4.1. Since here the purpose is to evaluate the effectiveness of CAGC-based message-passing, we only compare with baselines that focus on graph convolution (besides the classic MF) including the state-of-the-art GNN-based recommendation models (i.e., UltraGCN and GTN). Note that our work could be further enhanced if incorporating other techniques such as contrastive learning to derive self-supervision but stacking these would sidetrack the main topic of this paper, graph convolution, so we leave them as one future direction.

4.1.3 CAGCN-variants. For CAGCN, we calculate the edge weight solely based on our proposed CIR in message-passing by setting $g(A, B) = A$ in Eq. (7) and set $\gamma_i = \sum_{r \in \mathcal{N}_i^1} d_i^{-0.5} d_r^{-0.5}$ to ensure that the total edge weights for messages received by each node are the same as the one in LightGCN. For CAGCN*, we set g as the weighted summation and set $\gamma_i = \gamma$ as a constant controlling the trade-off between contributions from message-passing by LightGCN and by CAGC. We term the model variant as CAGCN(*)-jc if we use Jaccard Similarity (JC) [17] to compute $\mathbf{\Phi}$. The same rule applies to other topological metrics listed in Appendix A.1. Concrete equations of CAGCN and CAGCN* are provided in Appendix A.4.2.

4.1.4 Evaluation Metrics. Two popular metrics: Recall and Normalized Discounted Cumulative Gain(NDCG) [37] are adopted for evaluation. We set the default value of K as 20 and report the average of Recall@20 and NDCG@20 over all users in the test set. During inference, we treat items that the user has never interacted with in the training set as candidate items. All models predict users' preference scores over these candidate items and rank them based on the computed scores to further calculate Recall@20 and NDCG@20.

4.2 Performance Comparison

We first compare our proposed CAGCN-variants with LightGCN. In Table 2, CAGCN-jc/sc/lhn achieves higher performance than LightGCN because we aggregate more information from nodes with higher CIR(jc, sc, lhn) that bring more beneficial collaborations as justified in Section 2.2. CAGCN-cn generally performs worse than LightGCN because nodes having more common neighbors with other nodes tend to have higher degrees and blindly aggregating information more from these nodes would cause false-positive link prediction. Since different datasets exhibit different patterns of 2^{nd}-order connectivity, there is no fixed topological metric that performs the best among all datasets. For example, CAGCN-jc performs better than CAGCN-sc on Yelp and News, while worse on Gowalla, Ml-1M.

Table 3: Performance comparison of CAGCN* with GTN.

Model	Metric	GTN	CAGCN*		
			-jc	-sc	-lhn
Gowalla	Recall@20	0.1870	**0.1901**	0.1899	0.1885
	NDCG@20	0.1588	**0.1604**	0.1603	0.1576
Yelp2018	Recall@20	0.0679	**0.0731**	0.0729	0.0689
	NDCG@20	0.0554	**0.0605**	0.0601	0.0565
Amazon	Recall@20	0.0450	0.0573	**0.0575**	0.0520
	NDCG@20	0.0346	0.0456	**0.0458**	0.0409

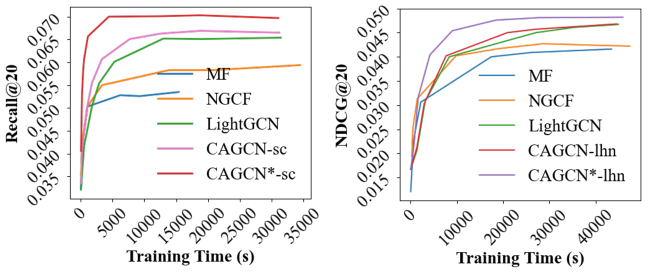
Then, we compare CAGCN*-variants with other baselines. We omit CAGCN*-cn here due to the worse performance of CAGCN-cn than LightGCN. We can see that CAGCN*-jc/sc almost consistently achieves higher performance than other baselines except for UltraGCN on Amazon. This is because UltraGCN allows multiple negative samples for each positive interaction, e.g., 500 negative samples here on Amazon[5], which lowers the efficiency as we need to spend more time preparing a large number of negative samples per epoch. Among the baselines, UltraGCN exhibits the strongest performance because it approximates the infinite layers of message passing and constructs the user-user graphs to capture 2^{nd}-order connectivity. LightGCN and NGCF perform better than MF since they inject the collaborative effect directly through message-passing.

To align the setting with GTN, we increase the embedding size d^0 to 256 following [4][6] and observe the consistent superiority of our model over GTN in Table 3. This is because in GTN [4], the edge weights for message-passing are still computed based on node embeddings that implicitly encode noisy collaborative signals from unreliable interactions. Conversely, our CAGCN* directly alleviates the propagation on unreliable interactions based on its CIR value, which removes noisy interactions from the source.

[5] UltraGCN negative samples: 1500/800/500/200 on Gowalla/Yelp2018/Amazon/Ml-1M.
[6] As the user/item embedding is a significant hyperparameter, it is crucial to ensure the same embedding size when comparing models; thus, we separately compare against GTN using their larger embedding size.

Table 4: Efficiency comparison of CAGCN* with LightGCN. For fair comparison, we track the first time CAGCN* achieves the best performance of LightGCN

Model	Stage	Gowalla	Yelp	Amazon	Ml-1M	Loseit	News
LightGCN	Training	16432.0	28788.0	81976.5	18872.3	39031.0	13860.8
CAGCN*	Preprocess	167.4	281.6	1035.8	33.8	31.4	169.0
	Training	2963.2	1904.4	1983.9	11304.7	10417.7	1088.4
	Total	3130.6	2186.0	3019.7	11338.5	10449.1	1157.4
Improve	Training	82.0%	93.4%	97.6%	40.1%	73.3%	92.1%
	Total	80.9%	92.4%	96.3%	39.9%	73.2%	91.6%



(a) Yelp (a) Loseit

Figure 5: Training time (s) of different models.

4.3 Efficiency Comparison

As recommendation models will be eventually deployed in user-item data of real-world scale, it is crucial to compare the efficiency of the proposed CAGCN(*) with other baselines. To guarantee a fair comparison, we use a uniform code framework implemented ourselves for all models and run them on the same machine with Ubuntu 20.04 system, AMD Ryzen 9 5900 12-Core Processor (3.0 GHz), 128 GB RAM and GPU NVIDIA GeForce RTX 3090. We report the Recall@20 on Yelp and NDCG@20 on Loseit achieved by the best CAGCN(*) variant based on Table 2. We track the performance and the training time per 5 epochs. Complete results are included in Supplementary B.5. In Figure 5(a)-(b), CAGCN achieves higher performance than LightGCN in less time. We hypothesize that for each user, its neighbors with higher interactions with its whole neighborhood would also have higher interactions with its interacted but unobserved neighbors. Then as CAGCN aggregate more information from these observed neighbors that have higher interactions with the whole neighborhood, it indirectly enables the user to aggregate more information from its to-be-predicted neighbors.

To verify the above hypothesis, we define the to-be-predicted neighborhood set of user u in the testing set as $\widehat{\mathcal{N}}_u^1$ and for each neighbor $j \in \mathcal{N}_u^1$, calculate its CIR $\widehat{\phi}_u^L(j)$ with nodes in $\widehat{\mathcal{N}}_u^1$. Then we compare the ranking consistency among CIRs calculated from training neighborhoods (i.e., $\phi_u(j)$), from testing neighborhoods (i.e., $\widehat{\phi}_u(j)$) and from full neighborhoods (we replace $\widehat{\mathcal{N}}_u^1$ with $\mathcal{N}_u^1 \cup \widehat{\mathcal{N}}_u^1$ in Eq. (5)). Here we respectively use four topological metrics (JC, SC, LHN, and CN) to define f and rank the obtained three lists. Then, we measure the similarity of the ranked lists between Train-Test and between Train-Full by Rank-Biased Overlap (RBO) [42]. The averaged RBO values over all nodes $v \in \mathcal{V}$ on three datasets are shown in Table 5. It is clear that the RBO values on all these datasets are beyond 0.5, which verifies our hypothesis. The RBO value between Train-Full is always higher than the one between Train-Test because most interactions are in the training set.

Table 5: Average Rank-Biased Overlap (RBO) of the ranked neighbor lists between training (i.e., $\mathcal{N}_u^1$) and testing/full (i.e., $\widehat{\mathcal{N}}_u^1$ and $\mathcal{N}_u^1 \cup \widehat{\mathcal{N}}_u^1$, respectively) dataset over all nodes $u \in \mathcal{U}$

Metric	Gowalla		Yelp		Ml-1M	
	Train-Test	Train-Full	Train-Test	Train-Full	Train-Test	Train-Full
JC	0.604±0.129	0.902±0.084	0.636±0.124	0.897±0.081	0.848±0.092	0.978±0.019
SC	0.611±0.127	0.896±0.084	0.657±0.124	0.900±0.077	0.876±0.077	0.983±0.015
LHN	0.598±0.121	0.974±0.036	0.578±0.100	0.976±0.029	0.845±0.082	0.987±0.009
CN	0.784±0.120	0.979±0.029	0.836±0.100	0.983±0.023	0.957±0.039	0.995±0.006

Moreover, by combining two views of propagations, one from CAGC and one from LightGCN, CAGCN* achieves even higher performance with even less time. This is because keeping aggregating more information from neighbors with higher CIR (as CAGCN does) would prevent each user from aggregating information from his/her other neighbors. In addition, we report the first time that our best CAGCN* variant achieves the best performance of LightGCN on each dataset in Table 4. We also report the preprocessing time for pre-calculating the CIR matrix $\mathbf{\Phi}$ for our model to avoid any bias. We could see that even considering the preprocessing time, it still takes significantly less time for CAGCN* to achieve the same best performance as LightGCN, which highlights the broad prospects to deploy CAGCN* in real-world recommendations.

4.4 Further Probe

4.4.1 Performance grouped by node degrees. Here we group nodes by degree and visualize the average performance of each group. Comparing non-graph-based models (e.g., MF), graph-based models (e.g., LightGCN, CAGCN(*)) achieve higher performance for lower degree nodes $[0, 300)$ while lower performance for higher degree nodes $[300, \text{Inf})$. Since node degree follows the power-law distribution [32], the average performance of graph-based models is still higher than MF. On one hand, graph-based models leverage neighborhood to augment the weak supervision for low-degree nodes. On the other hand, they introduce noisy interactions for higher-degree nodes. It is also interesting to see the opposite performance trends under different evaluation metrics: NDCG prefers high-degree nodes while recall prefers low-degree nodes. This indicates that different evaluation metrics have different sensitivity to node degrees and an unbiased node-centric evaluator is desired.

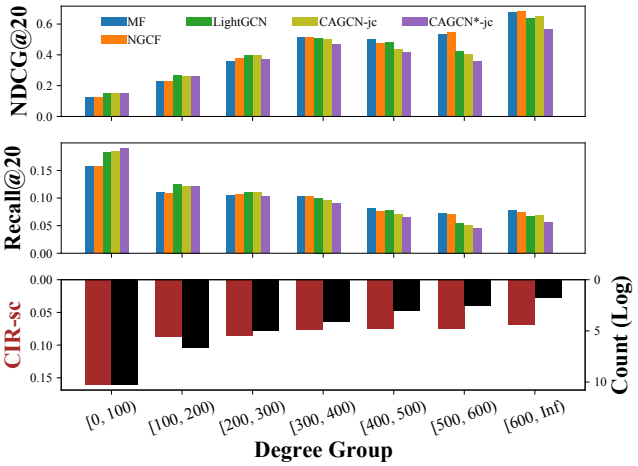
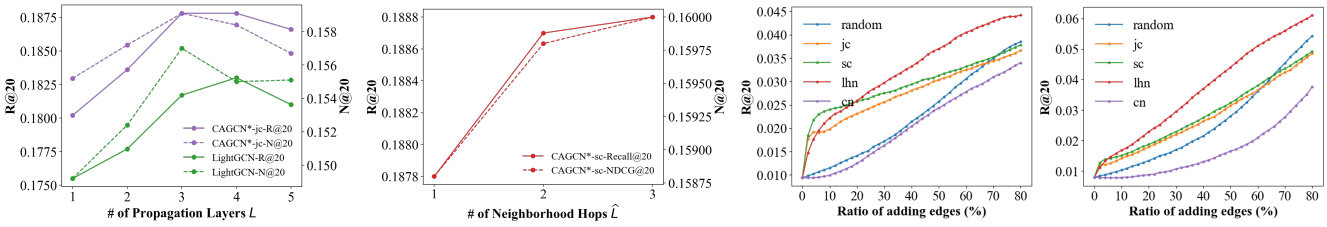


Figure 6: Performance w.r.t. node degree on Gowalla. A similar trend is seen on Yelp in Supplementary B.2.



(a) Gowalla - L (b) Gowalla - $\widehat{L}$ (c) Amazon (d) Yelp2018

Figure 7: In (a)-(b), the performance first increases since we capture higher-layer neighborhood information and higher-hop topological interaction in calculating CIR as $L, \widehat{L}$ increase from 1 to 3. However, the performance decreases in (a) as L increases due to over-smoothing. In (c)-(d), we add the global top edges directly (rather than cycle each node) according to their CIR. More details are provided in Appendix A.5.2.

4.4.2 Impacts of propagation layers L and neighborhood hops $\widehat{L}$. Figure 7(a)-(b) visualize the performance of CAGCN* and LightGCN when the propagation layer L in Eq. (2) and the neighborhood hop $\widehat{L}$ in Eq. (5) increase. In (a), the performance first increases as L increases from 1 to 3 due to the incorporation of high-layer neighborhood information and then decreases due to over-smoothing. More importantly, our CAGCN* is always better than LightGCN at all propagation layers. In (b), the performance consistently increases as the number of neighborhood hops increases because we are allowed to consider even more higher topological interactions among each node's neighborhood in computing CIR.

4.4.3 Adding edges globally according to CIR.. Figure 7(c)-(d) visualize the performance change when we add edges randomly and according to CIR. Unlike Figure 2-3 where we add edges by cycling each node, here we directly select the global top edges regardless of each center node according to their CIR and then evaluate the LightGCN with the pre-trained user-item embeddings. In the first stage, we observe a similar trend that adding edges according to JC, SC, and LHN leads to faster performance gain. However, since we don't cycle over each node, we would keep adding so many edges with larger CIR to the same node, which fails to bring performance gain anymore and hence cannot maximize our performance benefit under the node-centric evaluation metric.

5 RELATED WORK

Collaborative Filtering & Recommendation. Collaborative filtering (CF) predicts users' interests by utilizing the preferences of other users with similar interests [5]. Early CF methods used Matrix Factorization techniques [14, 26, 27, 33] to capture CF effect via optimizing users/items' embeddings over historical interactions. Stepping further, Graph-based methods either leverage topological constraints or message-passing to inject the CF effect into user/item embeddings [9, 37]. ItemRank and BiRank [6, 10] perform label propagation and compute users' ranking based on structural proximity between the observed and the target items. To make user preferences learnable, HOP-Rec [46] combines the graph-based method and the embedding-based method. Yet, interactions captured by random walks there do not fully explore the high-layer neighbors and multi-hop dependencies [39]. By contrast, GNN-based methods are superior at encoding higher-order structural proximity in user/item embeddings [9, 37]. Recent work [1, 4, 34] has demonstrated that not all captured collaborations improve users' ranking. [1] proposes to learn binary mask and impose low-rank regularization while ours propose novel topological metric CIR to weigh neighbors' importance. [4] smooths nodes' embeddings based on degree-normalized embedding similarity, while ours adaptively smooth based on topological proximity(CIR). [34] denoises interactions/preserve diversity based on 1-layer propagated embeddings and hence cannot go beyond 1-WL test, while ours keep neighbors and does not focus on diversity issues.

Link Prediction. As a generalized version of recommendation, link prediction finds applications in predicting drug interactions and completing knowledge graphs [22, 29]. Early studies adopt topological heuristics to score node pairs [15, 21, 52]. Furthermore, latent-based/deep-learning methods [25, 48] are proposed to characterize underline topological patterns in node embeddings via random walks [7] or regularizing [25]. To fully leverage node features, GNN-based methods are proposed and achieve unprecedented success owing to the use of the neural network to extract task-related information and the message-passing capture the topological pattern [23, 49, 51]. Recently, efforts have been invested in developing expressive GNNs that can go beyond the 1-WL test [19, 43, 50] for node/graph classification. Following this line, our work develops a recommendation-tailored graph convolution with provably expressive power in predicting links between users and items.

6 CONCLUSION

In this paper, we find that the message-passing captures collaborative effect by leveraging interactions between neighborhoods. The strength of the captured collaborative effect depends the embedding similarity, the weight of paths and the contribution of each propagation layer. To determine whether the captured collaborative effect would benefit the prediction of user preferences, we propose the Common Interacted Ratio (CIR) and empirically verify that leveraging collaborations from neighbors with higher CIR contributes more to users' ranking. Furthermore, we propose CAGCN(*) to selectively aggregate neighboring nodes' information based on their CIRs. We further define a new type of isomorphism, bipartite-subgraph-isomorphism, and prove that our CAGCN* can be more expressive than 1-WL in distinguishing subtree(subgraph)-isomorphic yet non-bipartite-subgraph-isomorphic graphs. Experimental results demonstrate the advantages of the proposed CAGCN(*) over other baselines. Specifically, CAGCN* outperforms the most representative graph-based recommendation model, LightGCN [9], by around 10% in Recall@20 but also achieves roughly more than 80% speedup. In the future, we will explore the imbalanced performance improvement among nodes in different degree groups as seen in Figure 6, especially from the perspective of GNN fairness [38, 41].

REFERENCES

[1] Huiyuan Chen, Lan Wang, Yusan Lin, Chin-Chia Michael Yeh, Fei Wang, and Hao Yang. 2021. Structured graph convolutional networks with stochastic masks for recommender systems. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 614–623.

[2] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM conference on recommender systems*. 191–198.

[3] Travis Ebesu, Bin Shen, and Yi Fang. 2018. Collaborative memory network for recommendation systems. In *The 41st international ACM SIGIR conference on research & development in information retrieval*. 515–524.

[4] Wenqi Fan, Xiaorui Liu, Wei Jin, Xiangyu Zhao, Jiliang Tang, and Qing Li. 2022. Graph Trend Filtering Networks for Recommendation. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 112–121.

[5] David Goldberg, David Nichols, Brian M Oki, and Douglas Terry. 1992. Using collaborative filtering to weave an information tapestry. *Commun. ACM* 35, 12 (1992), 61–70.

[6] Marco Gori, Augusto Pucci, V Roma, and I Siena. 2007. Itemrank: A random-walk based scoring algorithm for recommender engines.. In *IJCAI*, Vol. 7. 2766–2771.

[7] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. 855–864.

[8] Ruining He and Julian McAuley. 2016. VBPR: visual bayesian personalized ranking from implicit feedback. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 30.

[9] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yongdong Zhang, and Meng Wang. 2020. Lightgcn: Simplifying and powering graph convolution network for recommendation. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*. 639–648.

[10] Xiangnan He, Ming Gao, Min-Yen Kan, and Dingxian Wang. 2016. Birank: Towards ranking on bipartite graphs. *IEEE Transactions on Knowledge and Data Engineering* 29, 1 (2016), 57–71.

[11] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web*. 173–182.

[12] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*.

[13] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. 2019. Predict then Propagate: Graph Neural Networks meet Personalized PageRank. In *7th International Conference on Learning Representations, ICLR*.

[14] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer* 42, 8 (2009), 30–37.

[15] Elizabeth A Leicht, Petter Holme, and Mark EJ Newman. 2006. Vertex similarity in networks. *Physical Review E* 73, 2 (2006), 026120.

[16] Xin Li and Hsinchun Chen. 2013. Recommendation as link prediction in bipartite graphs: A graph kernel-based machine learning approach. *Decision Support Systems* 54, 2 (2013), 880–890.

[17] David Liben-Nowell and Jon Kleinberg. 2007. The link-prediction problem for social networks. *Journal of the American society for information science and technology* 58, 7 (2007), 1019–1031.

[18] Meng Liu, Hongyang Gao, and Shuiwang Ji. 2020. Towards deeper graph neural networks. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*. 338–348.

[19] Meng Liu, Haiyang Yu, and Shuiwang Ji. 2022. Your Neighbors Are Communicating: Towards Powerful and Scalable Graph Neural Networks. *arXiv preprint arXiv:2206.02059* (2022).

[20] Kelong Mao, Jieming Zhu, Xi Xiao, Biao Lu, Zhaowei Wang, and Xiuqiang He. 2021. UltraGCN: Ultra Simplification of Graph Convolutional Networks for Recommendation. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. 1253–1262.

[21] Mark EJ Newman. 2001. Clustering and preferential attachment in growing networks. *Physical review E* 64, 2 (2001), 025102.

[22] Maximilian Nickel, Kevin Murphy, Volker Tresp, and Evgeniy Gabrilovich. 2015. A review of relational machine learning for knowledge graphs. *Proc. IEEE* 104, 1 (2015), 11–33.

[23] Liming Pan, Cheng Shi, and Ivan Dokmanić. 2021. Neural Link Prediction with Walk Pooling. *arXiv preprint arXiv:2110.04375* (2021).

[24] Dongmin Park, Hwanjun Song, Minseok Kim, and Jae-Gil Lee. 2020. TRAP: Two-level regularized autoencoder-based embedding for power-law distributed data. In *Proceedings of The Web Conference 2020*. 1615–1624.

[25] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. 701–710.

[26] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian personalized ranking from implicit feedback. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*. 452–461.

[27] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2012. BPR: Bayesian personalized ranking from implicit feedback. *arXiv preprint arXiv:1205.2618* (2012).

[28] Francesco Ricci, Lior Rokach, and Bracha Shapira. 2011. Introduction to recommender systems handbook. In *Recommender systems handbook*. Springer.

[29] Benedek Rozemberczki, Charles Tapley Hoyt, Anna Gogleva, Piotr Grabowski, Klas Karis, Andrej Lamov, Andriy Nikolov, Sebastian Nilsson, Michael Ughetto, Yu Wang, et al. 2022. ChemicalX: A Deep Learning Library for Drug Pair Scoring. *arXiv preprint arXiv:2202.05240* (2022).

[30] Gerard Salton. 1989. Automatic text processing: The transformation, analysis, and retrieval of. *Reading: Addison-Wesley* 169 (1989).

[31] Walid Shalaby, Sejoon Oh, Amir Afsharinejad, Srijan Kumar, and Xiquan Cui. 2022. M2TRec: Metadata-aware Multi-task Transformer for Large-scale and Cold-start free Session-based Recommendations. In *Proceedings of the 16th ACM Conference on Recommender Systems*. 573–578.

[32] Andrew T Stephen and Olivier Toubia. 2009. Explaining the power-law degree distribution in a social commerce network. *Social Networks* 31, 4 (2009), 262–270.

[33] Yi Tay, Luu Anh Tuan, and Siu Cheung Hui. 2018. Latent relational metric learning via memory-based attention for collaborative ranking. In *WWW*. 729–739.

[34] Changxin Tian, Yuexiang Xie, Yaliang Li, Nan Yang, and Wayne Xin Zhao. 2022. Learning to Denoise Unreliable Interactions for Graph Collaborative Filtering. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 122–132.

[35] Chen Wang, Yuheng Qiu, Dasong Gao, and Sebastian Scherer. 2022. Lifelong graph learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 13719–13728.

[36] Junshan Wang, Guojie Song, Yi Wu, and Liang Wang. 2020. Streaming graph neural networks via continual learning. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 1515–1524.

[37] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. 2019. Neural graph collaborative filtering. In *Proceedings of the 42nd international ACM SIGIR conference on Research and development in Information Retrieval*. 165–174.

[38] Yu Wang. 2022. Fair Graph Representation Learning with Imbalanced and Biased Data. In *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*.

[39] Yu Wang and Tyler Derr. 2021. Tree Decomposed Graph Neural Network. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. 2040–2049.

[40] Yu Wang, Wei Jin, and Tyler Derr. 2022. Graph neural networks: Self-supervised learning. *Graph Neural Networks: Foundations, Frontiers, and Applications* (2022).

[41] Yu Wang, Yuying Zhao, Yushun Dong, Huiyuan Chen, Jundong Li, and Tyler Derr. 2022. Improving fairness in graph neural networks via mitigating sensitive attribute leakage. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 1938–1948.

[42] William Webber, Alistair Moffat, and Justin Zobel. 2010. A similarity measure for indefinite rankings. *ACM TOIS* 28, 4 (2010), 1–38.

[43] Asiri Wijesinghe and Qing Wang. 2021. A New Perspective on" How Graph Neural Networks Go Beyond Weisfeiler-Lehman?". In *ICLR*.

[44] Jiancan Wu, Xiang Wang, Fuli Feng, Xiangnan He, Liang Chen, Jianxun Lian, and Xing Xie. 2021. Self-supervised graph learning for recommendation. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 726–735.

[45] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2018. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826* (2018).

[46] Jheng-Hong Yang, Chih-Ming Chen, Chuan-Ju Wang, and Ming-Feng Tsai. 2018. HOP-rec: high-order proximity for implicit recommendation. In *Proceedings of the 12th ACM Conference on Recommender Systems*. 140–144.

[47] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. 2018. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD*. 974–983.

[48] Muhan Zhang and Yixin Chen. 2017. Weisfeiler-lehman neural machine for link prediction. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*. 575–583.

[49] Muhan Zhang and Yixin Chen. 2018. Link prediction based on graph neural networks. *Advances in neural information processing systems* 31 (2018).

[50] Lingxiao Zhao, Wei Jin, Leman Akoglu, and Neil Shah. 2021. From Stars to Subgraphs: Uplifting Any GNN with Local Structure Awareness. In *International Conference on Learning Representations*.

[51] Tong Zhao, Gang Liu, Daheng Wang, Wenhao Yu, and Meng Jiang. 2022. Learning from counterfactual links for link prediction. In *International Conference on Machine Learning*. PMLR, 26911–26926.

[52] Tao Zhou, Linyuan Lü, and Yi-Cheng Zhang. 2009. Predicting missing links via local information. *The European Physical Journal B* 71, 4 (2009), 623–630.

[53] Zixu Zhuang, Sheng Wang, Liping Si, Kai Xuan, Zhong Xue, Dinggang Shen, Lichi Zhang, Weiwu Yao, and Qian Wang. 2022. Local Graph Fusion of Multi-view MR Images for Knee Osteoarthritis Diagnosis. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer, 554–563.

A APPENDIX

A.1 Graph Topological Metrics for CIR

Here we demonstrate that by configuring different f and $\widehat{L}$, $\phi_u^{\widehat{L}}(j)$ can express many existing graph similarity metrics.

$$\phi_u^{\widehat{L}}(j) = \frac{1}{|\mathcal{N}_u^1|} \sum_{i \in \mathcal{N}_u^1} \sum_{l=1}^{\widehat{L}} \beta^{2l} \sum_{P_{ji}^{2l} \in \mathscr{P}_{ji}^{2l}} \frac{1}{f(\{\mathcal{N}_k^1 | k \in P_{ji}^{2l}\})} \tag{8}$$

- **Jaccard Similarity (JC) [17]:** The JC score measures the similarity between neighborhood sets as the ratio of the intersection of two neighborhood sets to the union of these two sets:

$$\mathrm{JC}(i,j) = \frac{|\mathcal{N}_i^1 \cap \mathcal{N}_j^1|}{|\mathcal{N}_i^1 \cup \mathcal{N}_j^1|} \tag{9}$$

Let $\widehat{L} = 1$ and set $f(\{\mathcal{N}_k^1 | k \in P_{ji}^2\}) = |\mathcal{N}_i^1 \cup \mathcal{N}_j^1|$, then we have:

$$\phi_u^1(j) = \frac{1}{|\mathcal{N}_u^1|} \sum_{i \in \mathcal{N}_u^1} \beta^2 \sum_{P_{ji}^2 \in \mathscr{P}_{ji}^2} \frac{1}{|\mathcal{N}_i^1 \cup \mathcal{N}_j^1|} = \frac{\beta^2}{|\mathcal{N}_u^1|} \sum_{i \in \mathcal{N}_u^1} \frac{|\mathcal{N}_i^1 \cap \mathcal{N}_j^1|}{|\mathcal{N}_i^1 \cup \mathcal{N}_j^1|} = \frac{\beta^2}{|\mathcal{N}_u^1|} \sum_{i \in \mathcal{N}_u^1} \mathrm{JC}(i,j) \tag{10}$$

- **Salton Cosine Similarity (SC) [30]:** The SC score measures the cosine similarity between the neighborhood sets of two nodes:

$$\mathrm{SC}(i,j) = \frac{|\mathcal{N}_i^1 \cap \mathcal{N}_j^1|}{\sqrt{|\mathcal{N}_i^1 \cup \mathcal{N}_j^1|}} \tag{11}$$

let $\widehat{L} = 1$ and set $f(\{\mathcal{N}_k^1 | k \in P_{ji}^2\}) = \sqrt{|\mathcal{N}_i^1 \cup \mathcal{N}_j^1|}$, then we have:

$$\phi_u^1(j) = \frac{1}{|\mathcal{N}_u^1|} \sum_{i \in \mathcal{N}_u^1} \beta^2 \sum_{P_{ji}^2 \in \mathscr{P}_{ji}^2} \frac{1}{\sqrt{|\mathcal{N}_i^1 \cup \mathcal{N}_j^1|}} = \frac{\beta^2}{|\mathcal{N}_u^1|} \sum_{i \in \mathcal{N}_u^1} \frac{|\mathcal{N}_i^1 \cap \mathcal{N}_j^1|}{\sqrt{|\mathcal{N}_i^1 \cup \mathcal{N}_j^1|}} = \frac{\beta^2}{|\mathcal{N}_u^1|} \sum_{i \in \mathcal{N}_u^1} \mathrm{SC}(i,j) \tag{12}$$

- **Common Neighbors (CN) [21]:** The CN score measures the number of common neighbors of two nodes and is frequently used for measuring the proximity between two nodes:

$$\mathrm{CN}(i,j) = |\mathcal{N}_i^1 \cap \mathcal{N}_j^1| \tag{13}$$

Let $\widehat{L} = 1$ and set $f(\{\mathcal{N}_k^1 | k \in P_{ji}^2\}) = 1$, then we have:

$$\phi_u^1(j) = \frac{1}{|\mathcal{N}_u^1|} \sum_{i \in \mathcal{N}_u^1} \beta^2 \sum_{P_{ji}^2 \in \mathscr{P}_{ji}^2} 1 = \frac{\beta^2}{|\mathcal{N}_u^1|} \sum_{i \in \mathcal{N}_u^1} |\mathcal{N}_i^1 \cap \mathcal{N}_j^1| = \frac{\beta^2}{|\mathcal{N}_u^1|} \sum_{i \in \mathcal{N}_u^1} \mathrm{CN}(i,j) \tag{14}$$

Since CN does not contain any normalization to remove the bias of degree in quantifying proximity and hence performs worse than other metrics as demonstrated by our recommendation experiments in Table 2.

- **Leicht-Holme-Nerman (LHN) [15]:** LHN is very similar to SC. However, it removes the square root in the denominator and is more sensitive to the degree of node:

$$\mathrm{LHN}(i,j) = \frac{|\mathcal{N}_i^1 \cap \mathcal{N}_j^1|}{|\mathcal{N}_i^1| \cdot |\mathcal{N}_j^1|} \tag{15}$$

Let $\widehat{L} = 1$ and set $f(\{\mathcal{N}_k^1 | k \in P_{ji}^2\}) = |\mathcal{N}_i^1| \cdot |\mathcal{N}_j^1|$, then we have:

$$\phi_u^1(j) = \frac{1}{|\mathcal{N}_u^1|} \sum_{i \in \mathcal{N}_u^1} \beta^2 \sum_{P_{ji}^2 \in \mathscr{P}_{ji}^2} \frac{1}{|\mathcal{N}_i^1| \cdot |\mathcal{N}_j^1|} = \frac{\beta^2}{|\mathcal{N}_u^1|} \sum_{i \in \mathcal{N}_u^1} \frac{|\mathcal{N}_i^1 \cap \mathcal{N}_j^1|}{|\mathcal{N}_i^1| \cdot |\mathcal{N}_j^1|} = \frac{\beta^2}{|\mathcal{N}_u^1|} \sum_{i \in \mathcal{N}_u^1} \mathrm{LHN}(i,j) \tag{16}$$

We further emphasize that our proposed CIR is a generalized version of these four existing metrics and can be delicately designed toward satisfying downstream tasks and datasets. We leave such exploration on the choice of f as one potential future work.

A.2 Derivation of Eq. (4)

The matrix form of computing the ranking of user u over item i after L-layer LightGCN-based message-passing:

$$y_{ui}^L = (\sum_{l_1=0}^{L} \beta_{l_1} \mathbf{E}_u^{l_1})^\top (\sum_{l_1=0}^{L} \beta_{l_1} \mathbf{E}_i^{l_1}) = (\sum_{l_1=0}^{L} \beta_{l_1} \mathbf{A}^{l_1} \mathbf{E}^0)_u^\top (\sum_{l_1=0}^{L} \beta_{l_1} \mathbf{A}^{l_1} \mathbf{E}^0)_i. \tag{17}$$

where β_{l_1} is the layer contribution and LightGCN uses mean-pooling, i.e., $\frac{1}{L}$ in Eq. (2). For the propagated embedding at a specific layer l_1, we have:

$$\mathbf{E}_u^{l_1} = (\mathbf{A}^{l_1} \mathbf{E}^0)_u = \sum_{j \in \mathcal{V}_u^{l_1}} \alpha_{ju}^{l_1} \mathbf{e}_j^0, \tag{18}$$

where $\alpha_{ju}^{l_1} = \sum_{P_{ju}^{l_1} \in \mathscr{P}_{ju}^{l_1}} \prod_{e_{pq} \in P_{ju}^{l_1}} d_p^{-0.5} d_q^{-0.5} (\alpha_{ju}^{l_1} = 0 \text{ if } \mathscr{P}_{ju}^{l_1} = \emptyset)$. $\mathcal{V}_u^{l_1}$ is the set of all nodes having paths of length l_1 to u and can be expressed as:

$$\mathcal{V}_u^{l_1} = \bigcup_{l_2=0}^{l_1} \mathcal{N}_u^{l_2} \cdot \mathbb{1}[(l_1 - l_2)\%2 = 0], \tag{19}$$

where

$$\mathcal{N}_u^{l_2} \cdot \mathbb{1}[(l_1 - l_2)\%2 = 0] = \begin{cases} \mathcal{N}_u^{l_2}, & (l_1 - l_2)\%2 = 0 \\ \emptyset, & (l_1 - l_2)\%2 \neq 0 \end{cases}. \tag{20}$$

Substituting Eq. (19) into Eq. (18), we have:

$$\mathbf{E}_u^{l_1} = (\mathbf{A}^{l_1} \mathbf{E}^0)_u = \sum_{j \in \mathcal{V}_u^{l_1}} \alpha_{ju}^{l_1} \mathbf{e}_j^0 = \sum_{j \in \bigcup_{l_2=0}^{l_1} \mathcal{N}_u^{l_2} \cdot \mathbb{1}[(l_1 - l_2)\%2 = 0]} \alpha_{ju}^{l_1} \mathbf{e}_j^0 = \sum_{l_2=0}^{l_1} \sum_{j \in \mathcal{N}_u^{l_2} \cdot \mathbb{1}[(l_1 - l_2)\%2 = 0]} \alpha_{ju}^{l_1} \mathbf{e}_j^0. \tag{21}$$

Then the aggregation of all L layers' embeddings of user u is expressed as:

$$\sum_{l_1=0}^{L} \beta_{l_1} \mathbf{E}_u^{l_1} = \sum_{l_1=0}^{L} \beta_{l_1} \sum_{l_2=0}^{l_1} \sum_{j \in \mathcal{N}_u^{l_2} \cdot \mathbb{1}[(l_1 - l_2)\%2]} \alpha_{ju}^{l_1} \mathbf{e}_j^0. \tag{22}$$

Eq. (22) means that for each length $l_1 \in \{0, 1, ..., L\}$, for each node $j \in \mathcal{V}_u^{l_1}$ that has path of length l_1 to u, we propagate its embedding over each path $P_{ju}^{l_1} \in \mathscr{P}_{ju}^{l_1}$ with the corresponding weight coefficient $\prod_{e_{pq} \in P_{ju}^{l_1}} d_p^{-0.5} d_q^{-0.5}$.

Since nodes that are l_1-hops away from u cannot have paths of length less than l_1, we reorganize Eq. (22) by first considering the hop of each node and then considering the length of each path, which leads to:

$$\sum_{l_1=0}^{L} \beta_{l_1} \mathbf{E}_u^{l_1} = \sum_{l_1=0}^{L} \beta_{l_1} \sum_{l_2=0}^{l_1} \sum_{j \in \mathcal{N}_u^{l_2} \cdot \mathbb{1}[(l_1 - l_2)\%2]} \alpha_{ju}^{l_2} \mathbf{e}_j^0 = \sum_{l_1=0}^{L} \sum_{j \in \mathcal{N}_u^{l_1}} \sum_{l_2=l_1}^{L} \beta_{l_2} \alpha_{ju}^{l_2} \mathbf{e}_j^0, \tag{23}$$

where $\alpha_{ju}^{l_2} = \sum_{P_{ju}^{l_2} \in \mathscr{P}_{ju}^{l_2}} \prod_{e_{pq} \in P_{ju}^{l_2}} d_p^{-0.5} d_q^{-0.5} (\alpha_{ju}^{l_2} = 0 \text{ if } \mathscr{P}_{ju}^{l_2} = \emptyset)$. Then by substituting Eq. (23) into Eq. (17), we end up with:

$$y_{ui}^L = (\sum_{l_1=0}^{L} \sum_{j \in \mathcal{N}_u^{l_1}} \sum_{l_2=l_1}^{L} \beta_{l_2} \alpha_{ju}^{l_2} \mathbf{e}_j^0)^\top (\sum_{l_1=0}^{L} \sum_{v \in \mathcal{N}_i^{l_1}} \sum_{l_2=l_1}^{L} \beta_{l_2} \alpha_{vi}^{l_2} \mathbf{e}_v^0), \tag{24}$$

where $\mathcal{N}_u^0 = \{u\}$ and specifically, $\alpha_{uu}^0 = 1$. β_{l_2} is the weight measuring contributions of propagated embeddings at layer l_2.

A.3 Complexity Comparison and Analysis

Let $|\mathcal{V}|, |\mathcal{E}|, |\mathcal{F}|$ be the total number of nodes, edges, and feature dimensions (assuming feature dimensions stay the same across all feature transformation layers). Let L be the propagation layer for all graph-based models using message-passing. Let r be the total number of negative samples per epoch per positive pair and K be the number of 2^{nd}-order neighbors. For r, all baselines use 1 per epoch per positive pair and hence can be omitted (aside from UltraGCN using a larger number). Then the complexity of each model is summarized in Table 6. For CAGCN, since we only consider 2-hops away connections to compute CIR in Eq. (5), the main computational load would be computing the power of adjacency matrix, which takes $O(|\mathcal{V}|^3)$. Note that for both of our CAGCN and UltraGCN, we can apply Strassens's Algorithm to further reduce the $O(|\mathcal{V}|^3)$ to $O(|\mathcal{V}|^{2.8})$. In Table 4 in Section 4.3, we report the preprocessing time for each dataset. Clearly, compared with the time used for training, the time for preprocessing is minor, which even demonstrates the superior efficiency of CAGCN since it significantly speeds up the training as justified in Section 4.3.

Table 6: Complexity of the pre-procession and the forward pass of CAGCN and different baselines.

Model		MF	NGCF	LightGCN
# Extra Hyper-parameters		/	/	1
Preprocess	Space	/	$O(\lvert\mathcal{E}\rvert + \lvert\mathcal{V}\rvert)$	$O(\lvert\mathcal{E}\rvert + \lvert\mathcal{V}\rvert)$
	Time	/	$O(\lvert\mathcal{E}\rvert + \lvert\mathcal{V}\rvert)$	$O(\lvert\mathcal{E}\rvert + \lvert\mathcal{V}\rvert)$
Training	Space	$O(\lvert\mathcal{V}\rvert F)$	$O(L\lvert\mathcal{V}\rvert F + \lvert\mathcal{E}\rvert + LF^2)$	$O(L\lvert\mathcal{V}\rvert F + \lvert\mathcal{E}\rvert)$
	Time	$O(\lvert\mathcal{E}\rvert F)$	$O(L(\lvert\mathcal{E}\rvert F + \lvert\mathcal{V}\rvert F^2))$	$O(L\lvert\mathcal{E}\rvert F + L\lvert\mathcal{V}\rvert F)$

Model		GTN	UltraGCN	CAGCN
# Extra Hyper-parameters		1	7	2
Preprocess	Space	$O(\lvert\mathcal{E}\rvert + \lvert\mathcal{V}\rvert)$	$O(\lvert\mathcal{E}\rvert + \lvert\mathcal{V}\rvert)$	$O(\lvert\mathcal{E}\rvert + \lvert\mathcal{V}\rvert)$
	Time	$O(\lvert\mathcal{E}\rvert + \lvert\mathcal{V}\rvert)$	$O(\lvert\mathcal{V}\rvert^3)$	$O(\lvert\mathcal{V}\rvert^3)$
Training	Space	$O(L\lvert\mathcal{V}\rvert F + \lvert\mathcal{E}\rvert)$	$O(\lvert\mathcal{V}\rvert F + \lvert\mathcal{V}\rvert K)$	$O(L\lvert\mathcal{V}\rvert F + \lvert\mathcal{E}\rvert)$
	Time	$O(L\lvert\mathcal{E}\rvert F + L\lvert\mathcal{V}\rvert F)$	$O(r(\lvert\mathcal{E}\rvert + \lvert V\rvert K)F)$	$O(L\lvert\mathcal{E}\rvert F + L\lvert\mathcal{V}\rvert F)$

A.4 Experimental Setting

A.4.1 Baselines. We compare our proposed CAGCN(*) with the following baselines: **MF [27]:** Most classic collaborative filtering method equipped with the BPR loss; **NGCF [37]:** The first GNN-based collaborative filtering model; **LightGCN [9]:** The most popular GNN-based collaborative filtering model, which removes feature transformation and nonlinear activation; **UltraGCN [20]:** The first model approximating regularization weights by infinite layers of message passing, and leveraging higher-order user-user relationships; **GTN [4]:** This model leverages a robust and adaptive propagation based on the trend of the aggregated messages to avoid unreliable user-item interactions.

A.4.2 CAGCN(*)-variants. For CAGCN, $\gamma_i = \sum_{r\in\mathcal{N}_i^1} d_i^{-0.5} d_r^{-0.5}$ to ensure that the total edge weights for messages received by each node are the same as LightGCN. Therefore, Eq. (7) becomes:

$$\mathbf{e}_i^{l+1} = \sum_{j\in\mathcal{N}_i^1} ((\sum_{r\in\mathcal{N}_i^1} d_i^{-0.5} d_r^{-0.5}) \frac{\mathbf{\Phi}_{ij}}{\sum_{k\in\mathcal{N}_i^1}\mathbf{\Phi}_{ik}}) \mathbf{e}_j^l, \forall i \in \mathcal{V}. \quad (25)$$

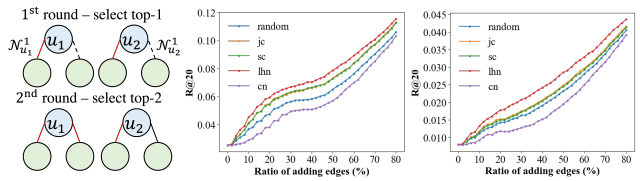
For CAGCN*, $\gamma_i = \gamma$ as a constant controlling the trade-off between contributions from message-passing according to LightGCN and according to CAGC. Eq. (7) becomes:

$$\mathbf{e}_i^{l+1} = \sum_{j\in\mathcal{N}_i^1} (\gamma \frac{\mathbf{\Phi}_{ij}}{\sum_{k\in\mathcal{N}_i^1}\mathbf{\Phi}_{ik}} + d_i^{-0.5} d_j^{-0.5}) \mathbf{e}_j^l, \forall i \in \mathcal{V}, \quad (26)$$

where we search γ in $\{1, 1.2, 1.5, 1.7, 2.0\}$.

A.5 Additional Experiments

A.5.1 Adding edges according to local CIRs. Given the user-item bipartite graph for training, we calculate the CIR-variants and use them to rank the neighborhood for each center node. During construction, we first remove all edges and then iteratively cycle over each node and add its corresponding neighbor based on the ranking until hitting the budget. Figure 8(a) contains an example with users u_1, u_2 and a budget of three edges, where u_1 and u_2 both first get an edge, but then only u_1 gets a second edge.

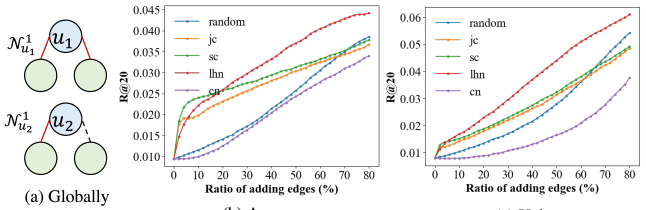


(a) Cycling node and adding edges (b) Gowalla (c) Yelp

Figure 8: (a) The procedure of adding edges according to CIR of neighbors around each node. (b)-(c) The performance change of adding edges on Gowalla and Yelp.

Similar to what we observed in Figure 3, the performance increases as we add more edges on Gowalla and Yelp (Figure 8(b) and (c), respectively). Furthermore, except for cn, adding edges according to CIR-variants is more effective in increasing the performance, which demonstrates the effectiveness of CIR in measuring the edge importance.

A.5.2 Adding edges according to global CIRs. Here we introduce how we add edges globally according to CIRs. Given the user-item interactions for training, we first construct the user-item bipartite graph and calculate the different variants of CIR including jc, sc, cn, lhn as stated in Appendix A.1. Then, we directly rank all edges according to the computed CIR. In the construction stage, we first remove all edges in the bipartite graph. Then we select the top edges according to the ranking based on our budget. Figure 9(a) contains an example with users u_1, u_2 and a budget of three edges, where we directly select the top-3 edges from all users' neighbors.



(a) Globally select top-3 (b) Amazon (c) Yelp

Figure 9: (a) The procedure of adding edges according to CIR globally. (b)-(c) The performance change of adding edges on Amazon and Yelp.

In the first stage, we observe a similar trend that adding edges according to CIRs lead to faster performance gain as Figure 8, which demonstrate the effectiveness of CIR in measuring the edge importance globally. However, since we don't cycle over each node and add its corresponding edge as we do in Appendix A.5.1, we would keep adding so many edges with larger CIR to the same node, which may not maximize our performance benefit when the metric is calculated by averaging over all nodes.

B SUPPLEMENTARY

B.1 Hyperparamters

We follow the procedure of hyperparameter tuning in [9, 37] and list the hyperparameters as follows:

- **LightGCN.** Propagation layers: $L = 3$; Pooling layer: Meaning pooling;
- **NGCF.** Propagation layers: $L = 3$; Slope of LeakyRelu: 0.2; Pooling layer: Concatenation
- **UltraGCN.** For Gowalla, Yelp, Amazon and Ml-1M, we use exactly the same hyperparameter configurations provided here. For Loseit and News, the hyperparamters are as follows:
  (1) Loseit: Training epochs 2000; Learning rate $1e^{-3}$; batch size 512; Loss weights $w_1 = 1e^{-6}, w_2 = 1, w_3 = 1e^{-6}, w_4 = 1$; the number of negative samples per epoch per positive pair 20; negative weight 20; weight of l_2 regularization $\gamma = 1e^{-4}$, 2^{nd}-constraining loss coefficient $\lambda = 5e^{-4}$.
  (2) News: Training epochs 2000; Learning rate $1e-3$; batch size 1024; Loss weights $w_1 = 1e^{-8}, w_2 = 1, w_3 = 1, w_4 = 1e^{-8}$; the number of negative samples per epoch per positive pair 1000; negative weight 200; weight of l_2 regularization $\gamma = 1e^{-4}$, 2^{nd}-constraining loss coefficient $\lambda = 5e^{-4}$.
- **GTN.** For Gowalla, Yelp, Amazon, we directly report the result provided here. In the following, we introduce the hyparamemeters we used for our CAGCN(*)-variants. With specification, the number of training epochs is set to be 1000; the learning rate 0.001; l_2 regularization $1e^{-4}$; number of negative samples 1; embedding dimenstion 64; batch size 256; $\widehat{L} = 1$.
- **CAGCN-jc.** (1) Gowalla: $\gamma = 1$; (2) Yelp: $\gamma = 1.2$; (3) Amazon: $\gamma = 1$; (4) Ml-1M: $\gamma = 2$; (5) Loseit: $\gamma = 1$; (6) News: $\gamma = 1$.
- **CAGCN-cn.** (1) Gowalla: $\gamma = 1$; (2) Yelp: $\gamma = 1.2$; (3) Amazon: $\gamma = 1$; (4) Ml-1M: $\gamma = 1$; (5) Loseit: $\gamma = 1$; (6) News: $\gamma = 1$.
- **CAGCN-sc.** (1) Gowalla: $\gamma = 1$; (2) Yelp: $\gamma = 1$; (3) Amazon: $\gamma = 1$; (4) Ml-1M: $\gamma = 2$; (5) Loseit: $\gamma = 1$; (6) News: $\gamma = 1$.
- **CAGCN-lhn.** (1) Gowalla: $\gamma = 1.2$; (2) Yelp: $\gamma = 1$; (3) Amazon: $\gamma = 1$; (4) Ml-1M: $\gamma = 2$; (5) Loseit: $\gamma = 1, L = 1$; (6) News: $\gamma = 1.5$.
- **CAGCN*-jc.** (1) Gowalla: $\gamma = 1.2$, l_2-regularization $1e-3$; (2) Yelp: $\gamma = 1.7$, l_2-regularization $1e^{-3}$; (3) Amazon: $\gamma = 1.7$, l_2-regularization $1e^{-3}$; (4) Ml-1M: $\gamma = 1$, l_2-regularization $1e^{-3}$; (5) Loseit: $\gamma = 1, L = 2$; (6) News: $\gamma = 1, L = 2$.
- **CAGCN*-sc.** (1) Gowalla: $\gamma = 1.2$, l_2-regularization $1e^{-3}$; (2) Yelp: $\gamma = 1.7$, l_2-regularization $1e^{-3}$; (3) Amazon: $\gamma = 1.7$, l_2-regularization $1e^{-3}$; (4) Ml-1M: $\gamma = 1$, l_2-regularization $1e^{-3}$; (5) Loseit: $\gamma = 1, L = 2$; (6) News: $\gamma = 1, L = 2$.
- **CAGCN*-lhn.** (1) Gowalla: $\gamma = 1$, l_2-regularization $1e^{-3}$; (2) Yelp: $\gamma = 1$, l_2-regularization $1e^{-3}$; (3) Amazon: $\gamma = 1.5$, l_2-regularization $1e^{-3}$; (4) Ml-1M: $\gamma = 1$, l_2-regularization $1e^{-3}$; (5) Loseit: $\gamma = 0.5, L = 2$; (6) News: $\gamma = 1, L = 2$.

B.2 Performance Interpretation

To demonstrate the generality of our observation in Figure 6, we further perform exactly the same analysis on Yelp (shown in Figure 10) and derive almost the same insights: 1) Graph-based recommendation models achieve higher performance than non-graph-based ones for lower degree nodes; 2) the opposite performance trends between NDCG and Recall indicates that different evaluation metrics have different levels of sensitivity to node degrees.

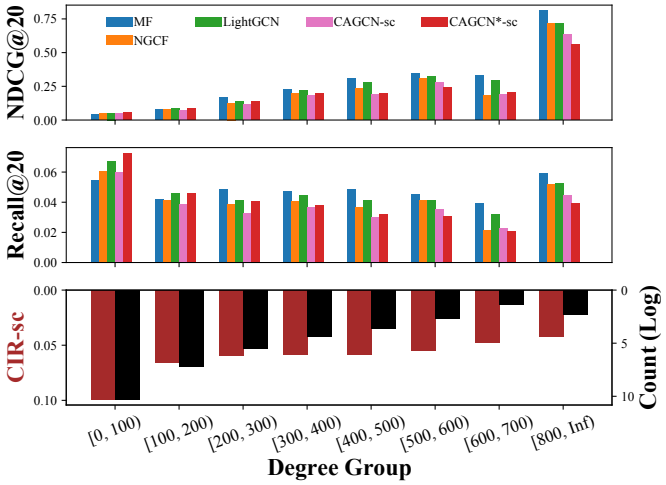


Figure 10: Performance of model w.r.t. node degree on Yelp.

B.3 Thorough Complexity Analysis

Generally compared with the very basic MF, the main computational issue of LightGCN comes from the message-passing which takes $O(L|\mathcal{E}|F)$ time and $O(L|\mathcal{V}|F)$ space to save the intermediate node representations. For NGCF, the extra complexity comes from the nonlinear transformation, which takes $O(L|\mathcal{V}|F^2)$ time and $O(LF^2)$ space to save the transformation weights. For UltraGCN, the main bottleneck comes from computing the user-user connections, which involves the power of adjacency matrix and hence $O(|\mathcal{V}|^3)$. Furthermore, as it samples hundreds of negative samples and the optimization is also performed on the user-user connections, then its time complexity would be $O(r(|\mathcal{E}| + |\mathcal{V}|K)F)$. For CAGCN, since we only consider 2-hops away connections to compute CIR in Eq. (5)(essentially for each center node, we count the number of paths of length 2 from each of its neighbors to its whole neighborhood), the main computational load would be computing the power of adjacency matrix, which takes $O(|\mathcal{V}|^3)$. Note that for both of our CAGCN and UltraGCN, we can apply Strassens's Algorithm to further reduce the $O(|\mathcal{V}|^3)$ to $O(|\mathcal{V}|^{2.8})$ for computing the power of adjacency matrix.

B.4 Graph Isomorphism

We review the concepts of subtree/subgraph-isomorphism [43].

Definition 3. Subtree-isomporphism: $\mathcal{S}_u$ and $\mathcal{S}_i$ are subtree-isomorphic, denoted as $\mathcal{S}_u \cong_{subtree} \mathcal{S}_i$, if there exists a bijective mapping $h : \widetilde{\mathcal{N}}_u^1 \to \widetilde{\mathcal{N}}_i^1$ such that $h(u) = i$ and $\forall v \in \widetilde{\mathcal{N}}_u^1, h(v) = j, \mathbf{e}_v^l = \mathbf{e}_j^l$.

Definition 4. Subgraph-isomporphism: $\mathcal{S}_u$ and $\mathcal{S}_i$ are subgraph-isomorphic, denoted as $\mathcal{S}_u \cong_{subgraph} \mathcal{S}_i$, if there exists a bijective mapping $h : \widetilde{\mathcal{N}}_u^1 \to \widetilde{\mathcal{N}}_i^1$ such that $h(u) = i$ and $\forall v_1, v_2 \in \widetilde{\mathcal{N}}_u^1, e_{v_1 v_2} \in \mathcal{E}_{\mathcal{S}_u}$ iff $e_{h(v_1)h(v_2)} \in \mathcal{E}_{\mathcal{S}_i}$ and $\mathbf{e}_{v_1}^l = \mathbf{e}_{h(v_1)}^l, \mathbf{e}_{v_2}^l = \mathbf{e}_{h(v_2)}^l$.

Corresponding to the backward($\Longleftarrow$) proof of Theorem 2, here we show two of such graphs $\mathcal{S}_u, \mathcal{S}'_u$, which are subgraph isomorphic but non-bipartite-subgraph-isomorphic. Assuming u and u' have exactly the same neighborhood feature vectors $\mathbf{e}$, then directly

propagating according to 1-WL or even considering node degree as the edge weight as GCN [12] can still end up with the same propagated feature for u and u'. However, if we leverage JC to calculate CIR as introduced in Appendix A.1, then we would end up with $\{(d_u d_{j_1})^{-0.5}\mathbf{e}, (d_u d_{j_2})^{-0.5}\mathbf{e}, (d_u d_{j_3})^{-0.5}\mathbf{e}\} \neq \{(d_{u'}^{-0.5} d_{j'_1}^{-0.5} + \widetilde{\Phi}_{u' j'_1})\mathbf{e}, (d_{u'}^{-0.5} d_{j'_2}^{-0.5} + \widetilde{\Phi}_{u' j'_2})\mathbf{e}, (d_{u'}^{-0.5} d_{j'_3}^{-0.5} + \widetilde{\Phi}_{u' j'_3})\mathbf{e}\}$. Since g is injective by Lemma 1, CAGCN would yield two different embeddings for u and u'.

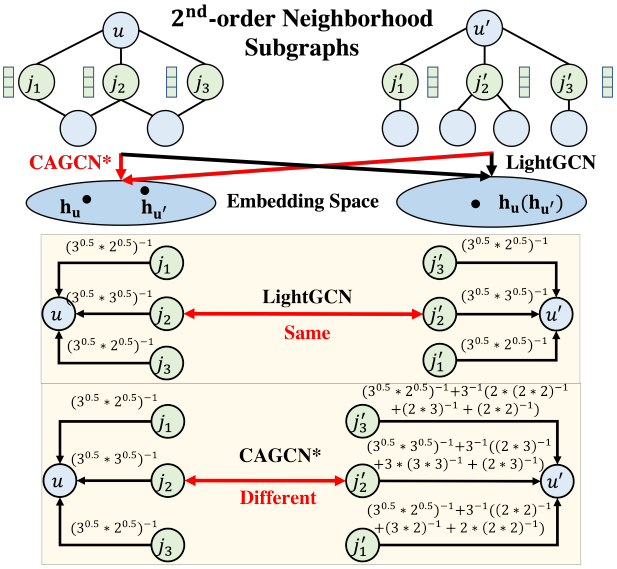


Figure 11: An example showing two neighborhood subgraph $\mathcal{S}_u, \mathcal{S}_{u'}$ that are subgraph-isomorphic but not bipartite-subgraph-isomorphic.

B.5 Efficiency Comparison

Here we use exactly the same setting introduced in Section 4.3 and keep track the performance/training time per 5 epochs for Gowalla, Yelp2018, Ml-1M, and Loseit in Figure 12. Clearly, CAGCN* achieves extremely higher performance in significantly less time because the collaboration-aware graph convolution leverages more beneficial collaborations from neighborhoods. Specifically, in Figure 12(c), we observe the slower performance increase of CAGCN* and LightGCN on Ml-1M. We ascribe this to the higher density of Ml-1M as in Table 1 that leads to so much noisy neighboring information. One future direction could be to leverage the CIR to prune the graph of these noisy connections in an iterative fashion as either a preprocessing step or even used throughout training when paired with an attention mechanism (although the latter would come at a significantly longer training time).

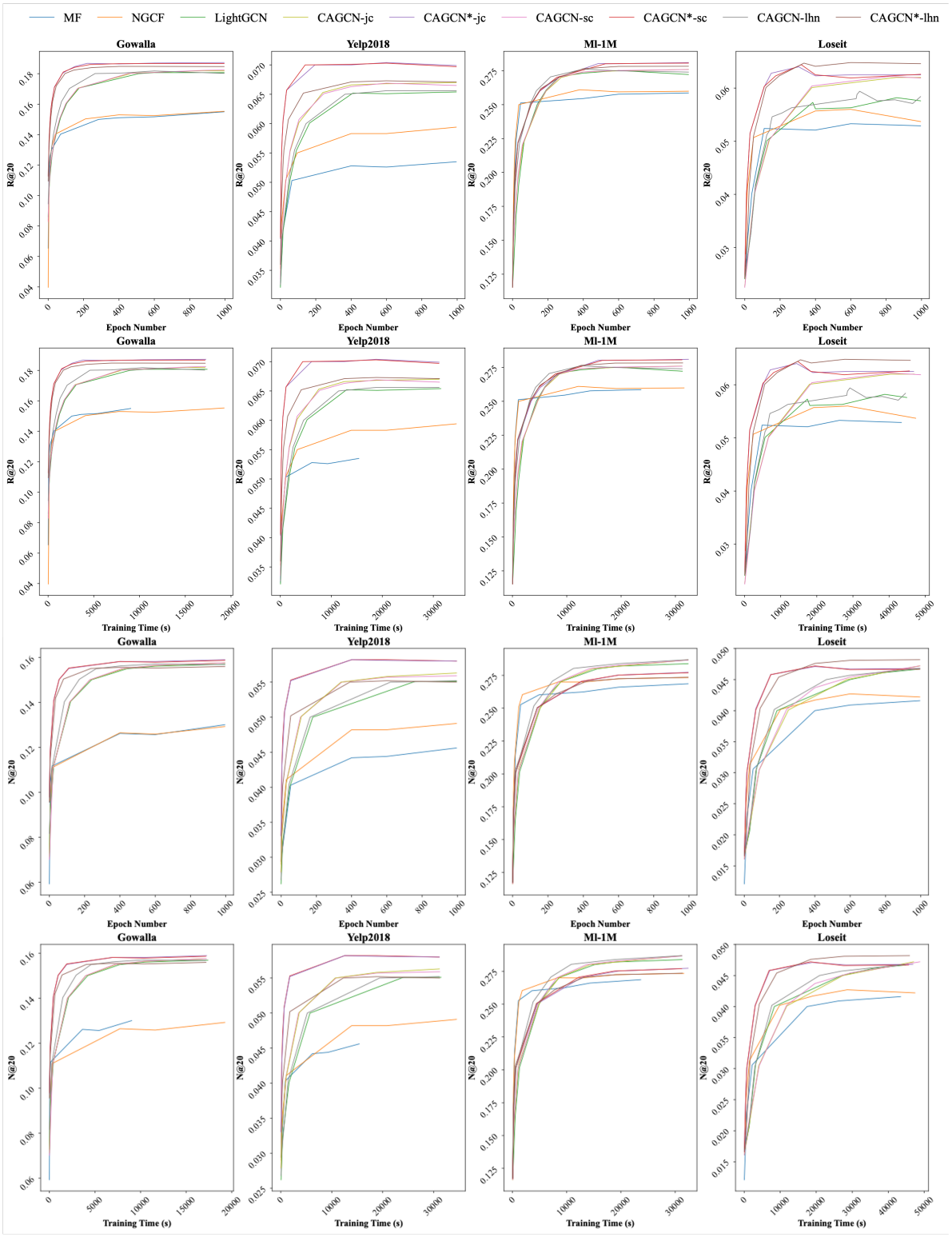


Figure 12: Comparing the training efficiency of each model under R@20 and N@20.