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.

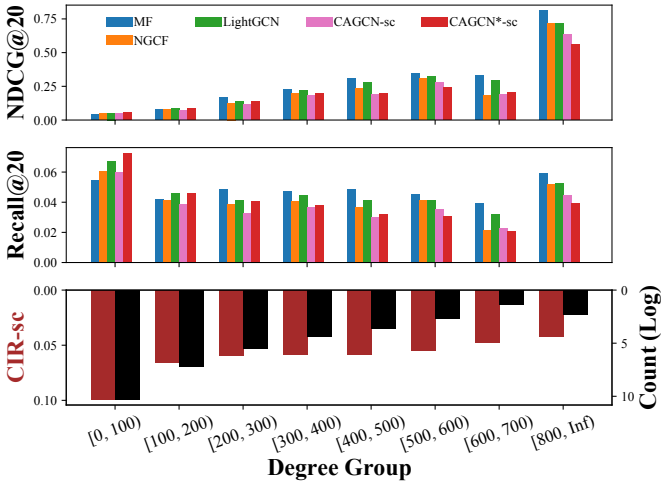

Figure 10: Performance of model w.r.t. node degree on Yelp.

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 $h : \widetilde{\mathcal{N}}_u^1 \rightarrow \widetilde{\mathcal{N}}_i^1$ 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 $h : \widetilde{\mathcal{N}}_u^1 \rightarrow \widetilde{\mathcal{N}}_i^1$ 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 [13] 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'.

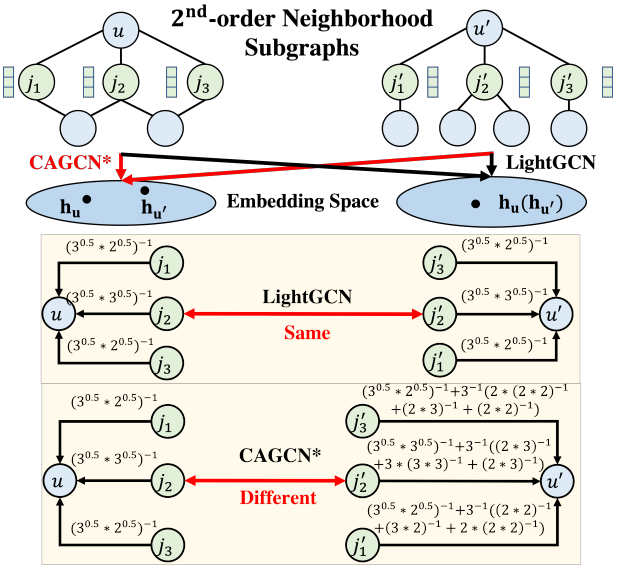

Figure 11: An example showing two neighborhood subgraph $\mathcal{S}_u, \mathcal{S}_{u'}$ that are subgraph-isomorphic but not bipartite-subgraph-isomorphic.

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).

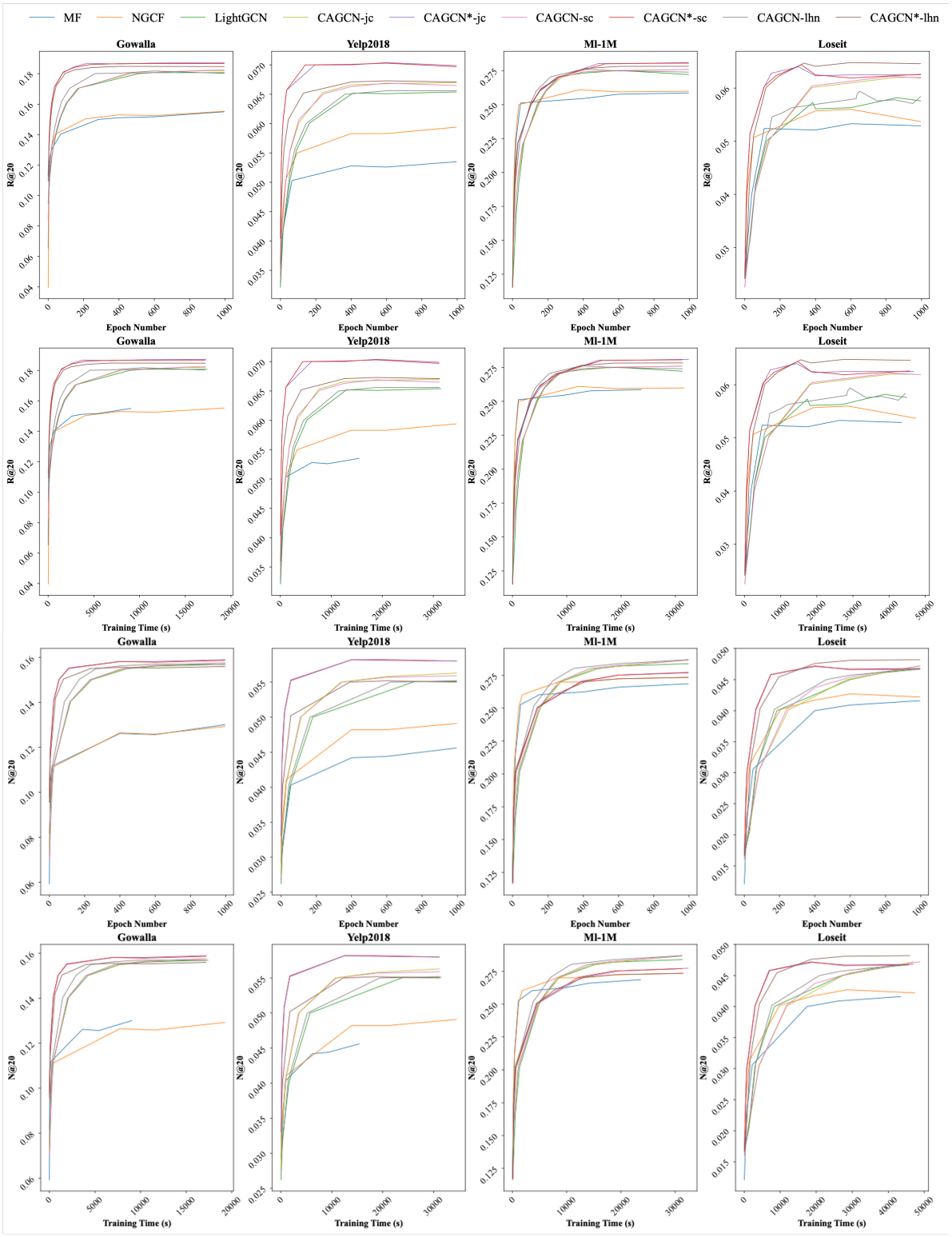

Figure 12: Comparing the training efficiency of each model under R@20 and N@20.