

**ERASMUS UNIVERSITY ROTTERDAM**

**Erasmus School of Economics**

**Master Thesis Data Science and Marketing Analytics**

**Dynamic Graph Networks for Sequential Recommender System  
with Personalised Subgraph Explanations: a Subgraph  
Isomorphism Counting Approach**

**Name student: Baichen Yan**

**Student ID number: 573475**

**Supervisor: Donkers, ACD**

**Second assessor: Wan, P**

**Date final version: 2023-10-02**

*The views stated in this thesis are those of the author and not necessarily those of the supervisor, second assessor, Erasmus School of Economics or Erasmus University Rotterdam.*

# Content

<b>1. Introduction</b>	<b>2</b>
1.1 Real-world Explanations for GNN in Recommender Systems	3
1.2 Dynamic GNN for Sequential Recommendations	3
1.3 Structure Dependency and Expressivity	4
1.4 Our Contributions	4
<b>2. Related Work</b>	<b>5</b>
2.1 Dynamic Graph Neural Network	6
2.2 Explainable Graph Neural Network	7
2.3 GNN-based Recommender System	8
2.4 Subgraph Isomorphism Counting	9
<b>3. Preliminaries</b>	<b>9</b>
<b>4. Methodology</b>	<b>11</b>
4.1 Graph Generation	13
4.1.1 Dynamic Graph Construction	13
4.1.2 Subgraph Sampling	14
4.2 Isomorphism Counting	15
4.2.1 Automorphism Orbit Counting	16
4.2.2 Subgraph Isomorphism Counting	18
4.3 Subgraph DGSR Mechanism	19
4.3.1 Information Encoding	20
4.3.2 Node Updating	21
4.3.3 Generate the Recommendations	22
4.3.4 Importance Scores	23
<b>5. Experiments</b>	<b>23</b>
5.1 Dataset	24
5.2 Model Settings	26
5.2.1 Default Parameter Set-up	26
5.2.2 Evaluation Methods	27
5.2.3 Environment & Parameter Setup	28
5.2.4 Subgraph List	28
RQ 1 Performance Comparison	29
RQ 2 Effect of Hyperparameters	33
RQ 3 Selection of Subgraphs	36
RQ 3.1 the number of subgraphs	36
RQ 3.2 the subgraph size	40
RQ 3.3 the ratio of users/items	42
<b>6. Case Study</b>	<b>45</b>
<b>7. Discussion</b>	<b>48</b>
<b>Reference</b>	<b>50</b>
<b>Appendix 1: Importance Scores of RQ1</b>	<b>55</b>
<b>Appendix 2: the Script of Subgraph isomorphism</b>	<b>56</b>

# 1. Introduction

As e-commerce platforms and online retailing services grow rapidly these years, recommender systems improve the relevance and quality of recommendations, expert at finding relevant and personalised recommendations for users by considering users' preferences, past behaviour, and demographic information. Various forms of recommender systems based on e-commerce platforms such as product suggestions websites Amazon and Taobao, or media services like Tiktok and Netflix. Undoubtedly, recommender systems have become indispensable tools to meet different business demands to boost sales (Covington, P. et al., 2016; Li, G. et al., 2020; Xie, X. et al., 2021; Ying, R. et al., 2018; Zhang, S. et al., 2018). Subsequently, a myriad of techniques has been developed to capture the features of consumers and products and to improve the recommender system. As a prominent, progressive, and powerful technology, the artificial neural networks are highly applied in the development of recommender systems (e.g. Covington, P., et al. 2016; Koren, Y., 2008).

Inspired by the powerful ability of Graph Neural Networks (GNN) learning on graph structure data, numerous research of GNN Recommender systems emerged (e.g. Chang, J., et al., 2021; Qiu, R. et al., 2019; Li, Z. et al., 2020). By leveraging the bipartite graph structure of user-item interactions, GNNs have been demonstrated to be powerful in representation learning and can be inherited by the applications to recommender systems (Wu, S. et al., 2022). The advantages include unifying structure data, capturing deep and complex relationships among multi-hop entities, and modelling high-order connectivity (He, X. et al., 2020; Ying, R. et al., 2018). In general, GNN are powerful models working on recommendation systems with impressive performance with a history of several years (Wu, S. et al., 2022).

With the progression of the GNN-based recommender system models, we observe three key problems remain in this field. In response, our research endeavours to make meaningful contributions towards advancing the understanding and resolution of these issues. Moreover, we observe the emergence of three distinct research streams, each dedicated to addressing these problems and presenting potential solutions. These research streams offer valuable insights to our research contribute to the ongoing evolution of knowledge in these problems:

## 1.1 Real-world Explanations for GNN in Recommender Systems

In contemporary explainable Graph Neural Network (GNN) models, according to Yuan's taxonomy (2022), the prevalent approaches focus on explaining instances or models, including highlighting influential features and generating importance scores, but often lack the proactive capability to select what to explain. These methods typically rely on pre-existing graph structures and we cannot select the structures we want to focus on. Besides, the importance scores or highlighted structures resulting from these approaches may lack meaningful interpretations and practical relevance in real-world scenarios, especially in e-commerce. The results may also provide implausible and unactionable advice for retailers, consequently reducing the practical usability of the model. For instance, perturbation-based models learn an algorithm to generate masks for processing input graphs and obtaining feedback from objective functions, but these masks heavily depend on the pre-existing subgraphs present in the data, which are hard to interpret to a retailing context.

To this end, our design is rooted in a unique approach: we track specific graph structures that emerge from actual user-item interaction cases. Our model traces these particular graph structures and searches within the data for instances with analogous patterns. By incorporating them into the model's input, we achieve a dual benefit. Not only does this elevate the model's performance significantly, but it also generates importance scores, enhancing its interpretability. This methodology revolves around transforming real purchase scenarios into graphical representations, enabling a more comprehensive capture of user behaviours and item relationships. This, in turn, optimises the recommendation process. Through this strategy, we not only enhance the quality of recommendations but also provide interpretability to each recommendation, allowing users to better understand the basis and rationale behind the suggestions.

## 1.2 Dynamic GNN for Sequential Recommendations

The second problem acknowledged a crucial constraint in the conventional GNN recommender system model, which predominantly captures static user-item interactions while overlooking the valuable insights offered by the history-rich sequential information of users (Zhang, M. et al., 2020; 2022). To overcome this limitation, various models were devised to incorporate historical data for sequential recommendations, including the Temporal Dependent GNN (Qu, L. et al., 2020) and the Graph Convolution embedded LSTM

(Chen, J. et al., 2022). Among these advancements, the Dynamic Graph Neural Network for Sequential Recommendations (DGSR) has gained considerable attention due to its specialised focus on sequential recommendations. Notably, DGSR effectively considers dynamic collaborative signals and high-order histories derived from user-item interactions (Zhang, M. et al., 2022). This emphasis on dynamic information has contributed significantly to its prominence in the field. DGSR places a strong emphasis on effectively processing dynamic collaborative signals of user sequences in the context of sequential recommendations, enhancing the accuracy and relevance of personalised suggestions. Throughout this research endeavour, DGSR takes priority as the primary model we follow and extensively investigate.

### 1.3 Structure Dependency and Expressivity

The last problem concentrates on the structure dependency and model expressivity. Some researchers notice that GNN models are unable to effectively capture long-range topological information from the underlying graph structure (Liu, J. et al., 2021). Even the dynamic GNN models encounter challenges when it comes to representing graph or subgraph structure patterns, such as denoising by neighbour sampling (Chu Y., et al., 2021). These GNN models also suffer from limited abilities (the Wefeiler Leman test) to adequately exploit graph substructure (Xu, K. et al., 2018; Morris, C. et al., 2019). A recent solution to this issue lies in Subgraph Isomorphism Counting (Bouritsas, G. et al., 2022). Bouritsas introduced an innovative message-passing scheme, referred to as Graph Substructure Network (GSN), which capitalises on the isomorphism counting metric. According to Bouritsas, GSN demonstrates strictly enhanced expressivity when compared to the conventional message-passing techniques employed in GNN models.

### 1.4 Our Contributions

Given the aforementioned challenges, we introduce a novel model called "Subgraph DGSR." Our approach is founded on the concept that purchasing behaviours tend to conform to patterns that are inherently shaped by user experiences and product-related factors. These patterns exert a natural influence on consumer decision-making. To illustrate, let's consider a straightforward pattern found in the purchase of a table and chairs. This purchase reflects a fundamental graph structure: item1-user-item2, signifying that users who buy item1 are likely to follow up with the purchase of item2. This particular scenario or pattern can occur numerous times across the entire dataset. Consequently, we aim to direct the model's attention

towards specific scenarios that frequently manifest in the data. This approach not only assists the model in extracting patterns more effectively from the data but also enables it to gauge the significance of these scenarios within the dataset.

Here are two key innovations in our model:

1. Our model accepts a flexible list of subgraphs input that enables researchers to tailor the subgraphs the model should focus on. These subgraphs can originate from real-world scenarios, offering deeper insights and handling complex patterns. For instance, consider a household scenario where the mother purchases a pet cage, the children buy pet toys, and the father buys pet food. These intricate cases, involving multiple users and items, can be challenging to incorporate into a traditional model. However, our model can accommodate such scenarios by leveraging isomorphism counting metrics to address this complexity;
2. Our model integrates knowledge from graph theory and isomorphism counting so that we don't need to exhaustively process every case in the form of one subgraph. Upon inputting a purchase case, our proposed model identifies other sequences of purchases with similar structure, taking into consideration the overall sales environment and measuring the contribution of these similar structures to the prediction.

In summary, our work makes the following contributions:

1. We introduce a novel structure-enhanced model "Subgraph DGSR", which requires a list of subgraphs as hyperparameters, representing specific patterns for the model to keep tracking, in order to enhance model performance and generate importance scores;
2. We demonstrate the performance of our proposed model through 2 evaluation methods on 3 real-world datasets in Amazon Review Data (Ni, J. et al., 2019);
3. We illustrate our explaining framework working with our model by tracing subgraph structures from real-world user-item cases and generating importance scores of them.

## 2. Related Work

In this section, we will provide a concise overview of prior research in four topics related our study: Dynamic Graph Neural Network (DGNN), Explainable Graph Neural Network (XGNN), Graph-based Recommender Systems, and Subgraph Isomorphism Counting.

## 2.1 Dynamic Graph Neural Network

Graph data frequently intersects temporal aspects across multiple applications, prompting the creation of a multitude of models designed to handle dynamic graphs in various domains, such as knowledge networks, social networks, and recommender systems. These models aim to capitalise on both the structural and temporal characteristics of the data simultaneously (Skarding, J. et al., 2021). Nevertheless, it's essential to note that not every model tailored for processing dynamic graphs can be classified as a DGNN. According to the definition proposed by Skarding (2021), a DGNN is specifically a deep representation learning model that integrates neighbouring nodes as an integral part of its neural architecture. This definition distinguishes models that depict dynamic graphs using deep techniques but do not inherently involve the aggregation of neighbouring nodes within their architecture.

For example, DANE utilises matrix perturbation theory to dynamically extract fluctuations in adjacency and attribute matrices (Li, J. et al., 2017). DynamicTriad introduces the concept of triadic closure to retain both structural details and evolving patterns within dynamic networks (Zhou, L. et al., 2018). DynGEM utilises an expanding deep AutoEncoder that adapts in real-time to capture intricate nonlinear relationships, including both first-order and second-order connections among graph nodes (Goyal, P. et al., 2018). HTNE incorporates the Hawkes process into network embeddings to grasp how previous neighbours influence current ones within temporal embedding (Zuo, Y. et al., 2018). Dyrep leverages a deep temporal point process model to represent temporally evolving structural information (Trivedi, R. et al., 2019). JODIE incorporates two types of recurrent neural networks (RNNs) to model the evolution of distinct node representations (Kumar, S. et al., 2019). MTNE not only integrates the Hawkes process to stimulate triadic evolution but also utilises attention mechanisms to discern the significance of various motifs (Huang, H. et al., 2020).

To enhance the models' capacity for capturing evolving graph structures, Xu, D. et al. (2020) have introduced a temporal graph attention mechanism grounded in Bochner's theorem. Additionally, certain research efforts have partitioned dynamic graphs into a series of separate graph snapshots. These advancements collectively underscore the persistent pursuit of comprehensive dynamic graph modelling, particularly when considering temporal aspects (Sankar, A. et al., 2020). Furthermore, in a sequential generation approach, DGRN

accentuates the dynamic collaborative signals during the encoding of user sequences (Zhang, M. et al., 2022).

## 2.2 Explainable Graph Neural Network

In recent years, the realm of explainable graph neural networks (XGNN) has gained prominence as a pivotal area of research, with the objective of providing users with clear and comprehensible insights into GNN models. Depending on various aspects of GNN encoding that are emphasised, XGNN can be categorised into three main types: XGNN through subgraphs, through graph generation, and through intermediate-level interventions (Li, Y. et al., 2022). Among these, the subgraph approach represents a family of techniques that leverage subgraphs to enhance the interpretability of GNN models. This approach frequently centres on local features, with a focus on identifying and presenting the most significant subgraph components.

As a pioneer in explaining GNN, GNNExplainer identifies a compact subgraph structure and a group of influential nodes in the prediction by highlighting relevant features and edges relevant to the predictions (Ying, Z. et al., 2019). Zhang et al. (2020) introduced RelEx, a model-agnostic explainer for relational models, by building a local differentiable approximation of the black-box model and generating an interpretable mask using subgraphs. This approach provides flexible explanations for users, with a locally faithful and differentiable approximator for GNN models regarding the input adjacency matrix.

Lin et al. (2020) introduced GISST, a model-agnostic framework for interpreting graph structures and node features by inducing sparsity, discarding unimportant elements. It identifies significant subgraphs and features through probability analysis of the adjacency matrix and node features matrix. Vu et al. (2020) proposed PGM-Explainer, which identifies crucial graph components to create Bayesian explanations, depicting feature dependencies and providing deep insights into GNN predictions. Yuan et al. (2021) developed SubgraphX, which utilises Shapley values to measure subgraph importance in GNNs, focusing on interactions within the information aggregation process. They employ the Monte Carlo tree search algorithm for efficient subgraph exploration, explicitly explaining GNNs through subgraph identification.



Despite the diverse approaches of the above-mentioned models for explaining subgraphs or features, none of them allows for the proactive selection of a specific subgraph to explain. In other words, they lack the capability to choose a particular subgraph for explanation.

## 2.3 GNN-based Recommender System

In the rapidly evolving landscape of recommender systems, Graph Neural Networks (GNNs) have emerged as powerful tools for capturing complex relationships in the scenarios of social network (Wu, L. et al., 2019), sequential (Chang, J. et al., 2021), session-based (Chen, T., & Wong, R. C. W., 2020), cross-domain (Guo, L. et al., 2021), multi-behavior (Jin, B. et al., 2020), bundle recommendation (Chang, J. et al., 2020).

Given the context of sequential recommendations (as we illuminate the importance of DGNN), the sequential behaviours are naturally expressed in the order of time. One main issue here could be the shorter and uncertain number of user/item sequences which reduce the capacity of GNN it utilises (Wu, S. et al., 2022). One mainstream way to solve this is to import more information into the original graph. HetGNN employs all behaviour sequences and establishes edges between two successive items within the same sequence, with each edge type representing a specific behaviour type (Wang, W. et al., 2020). A-PGNN focuses on scenarios where user information is available, integrating a user's historical sequences with the current sequence to enhance item-item connections (Zhang, M. et al., 2020). GCE-GNN (Wang, Z. et al., 2020) and DAT-MDI (Chen, C. et al., 2021) utilise item transitions from all sessions to capture transition patterns within the current sequence, leveraging both local and global contexts.

In the realm of sequential recommendations employing Graph Neural Networks (GNN), five prominent GNN frameworks are frequently employed: GCN approximates the first-order eigendecomposition of the graph Laplacian to iteratively gather information from neighbouring nodes (Kipf, T. N., & Welling, M., 2016). GraphSAGE samples a fixed-sized neighbourhood for each node, introduces mean/sum/max-pooling aggregators, and employs concatenation for updates (Hamilton, W. et al., 2017). GAT distinguishes the influence of neighbours, avoiding uniform or pre-defined influence assumptions, by using an attention mechanism to weigh the contributions of neighbouring nodes when updating a node's vector (Velickovic, P. et al., 2017). GGNN employs a gated recurrent unit (GRU) in the update step

(Li, Y. et al., 2015), and HGNN represents a typical hypergraph neural network, designed to capture high-order data correlations within a hypergraph structure (Feng, Y. et al., 2019).

## 2.4 Subgraph Isomorphism Counting

Isomorphism metrics have been utilised in graph-based models for a couple of years. For instance, they've been used to identify functional group information in chemical molecules for determining compound properties (Gilmer, J. et al., 2017), and in social networks, certain substructures have been considered crucial for recommender systems (Ying, R. et al., 2018).

While numerous graph-based models have emerged, many of them rely on multiple message-passing steps for nodes to comprehend the graph's global structure. However, these message-passing GNNs are limited to at most as powerful as the Weisfeiler Leman test (WL test, Weisfeiler, B., & Leman, A., 1968). Consequently, they are constrained in fully exploiting the graph structure (Morris et al., 2019; Xu et al., 2019).

Within the constraints, Bouritsas et al. (2022) introduced the Graph Substructure Network (GSN), a novel message-passing scheme that incorporates structural information into aggregation functions. Their theoretical analysis confirmed that GSN surpasses traditional GNN in expressiveness for most subgraph patterns while preserving localised message passing, setting it apart from higher-order methods like the Weisfeiler-Leman hierarchy (e.g., Morris, C. et al., 2019; Maron, H. et al., 2018, 2019 May, 2019).

GSN operates based on subgraph encoding and necessitates the selection of specific subgraphs for input. Notably, if these subgraph counts can uniquely identify vertices, GSN's universality property remains intact. Moreover, the authors conjecture that the number of subgraphs required for uniqueness can be significantly reduced in real-world scenarios.

## 3. Preliminaries

In this section, we define the symbols and outline the issues related to sequential graphs and isomorphism, paving the way for subsequent in-depth analysis and discussion.

### Sequential Graphs

In the context of bipartite graphs for sequential recommendations, the sets  $U$  and  $I$  denote users and items, respectively. For every user  $u \in U$ , their action sequence is represented as

$S^u = (i_1, i_2, \dots, i_k)$ , where  $i \in I$ . The corresponding timestamp sequence for  $S^u$  is denoted as  $T^u = (t_1, t_2, \dots, t_k)$ . The entire set of  $S^u$  is indicated as  $S$ . The objective of sequential recommendation is to predict the subsequent item in  $S^u$  using the sequence information between time  $t_1$  and  $t_k$ . Generally, the sequential recommendation task imposes a maximum length  $n$  on  $S^u$ . if  $k$  exceeds  $n$ , the prediction is made based on the most recent  $n$  items  $(i_{k-n}, i_{k-n+1}, \dots, i_k)$ . Each user and item is embedded into a low dimension vector  $e_u, e_i \in \mathbb{R}^d$ , where  $u \in U$  and  $i \in I$ , and  $d$  represents the dimension of the embedding space. The user embedding matrix is denoted as  $E_U \in \mathbb{R}^{|U| \times d}$ , and the item embedding matrix is represented as  $E_I \in \mathbb{R}^{|I| \times d}$ .

### Isomorphism and Orbit

Consider an undirected graph  $G = (V_G, E_G)$ , denoted as  $G = (V_G, E_G)$ , which is composed of a set of vertices  $V_G$  and a set of edges  $E_G$ . This graph is characterised by  $n$  and  $m$  edges. A subgraph of  $G$ , symbolised as  $G_S = (V_{G_S}, E_{G_S})$ , is any graph where  $V_{G_S} \subset V_G$  and  $E_{G_S} \subset E_G$ . An "induced" subgraph is defined when  $E_{G_S}$  encompasses all the edges of  $G$  that have endpoints in  $V_{G_S}$ , i.e.,  $E_{G_S} = E_G \cap (V_{G_S} \times V_{G_S})$ .

To prevent confusion with  $u$  representing users, we employ  $a$  or  $b$  to symbolise a vertex in a graph. Two graphs,  $G$  and  $H$ , are isomorphic (expressed as  $H \simeq G$ ), if there is a bijective mapping that preserves adjacency,  $f: V_G \rightarrow V_H$ . In other words,  $(b, a) \in E_G$  iff.  $(f(b), f(a)) \in E_H$ . In the context of a small graph  $H$ , the subgraph isomorphism problem involves identifying a subgraph  $G_S$  of  $G$  such that  $G_S \simeq H$ . An automorphism of  $H$  is an isomorphism that functions  $H$  to itself. The set of all automorphisms consists the automorphism group of the graph, denoted as  $Aut(H)$ , which contains all possible graph symmetries.

By  $Aut(H)$ , the vertices are separated into subsets of  $V_H$  called orbits. Formally, the orbit of a vertex  $v \in V_H$  is the set of vertices to which it can be mapped via an automorphism:  $Orb(b) = \{a \in V_H \mid \exists g \in Aut(H) s.t. g(a) = b\}$ . and the set of all orbits denoted as  $\{O_{H,1}^V, \dots, O_{H,d_H}^V\}$ , where  $d_H$  is the cardinality of the quotient. Similarly, structural roles of edges are defined through edge automorphisms, which are bijective mappings from the set of edges to themselves that maintain edge adjacency. Specifically, each vertex automorphism

$g$  induces an edge automorphism by mapping each edge  $(a, b)$  to  $(g(a), g(b))$ . Edge automorphism groups are formed in the same way, from which we derive the partition of the edge set into edge orbits  $\{O_{H,1}^E, \dots, O_{H,d_H}^E\}$ .

## 4. Methodology

In this section, we illustrate the model construction and delve into the methodology employed in developing our innovative recommendation system. The entire framework of our model is illustrated in Fig. 1. In the subsequent sections, we will explain the fundamental components of our model, accompanied by a detailed description of the approach employed to leverage subgraphs for personalised recommendations.

In Section 4.1, we detail the process of generating data graphs from user-item interaction data stored in the retailer's database. This process is thoroughly outlined in Algorithm 1 within this section.

In Section 4.2, we delve into the algorithm handling of the list of subgraphs that we want our model to be aware of. This algorithm enables our model to detect specified subgraph structures we input, independent of specific users and items, and ensures our model not to become overwhelmed by the multitude of users and items, maintaining focus only on the essential structures we define, and extracting structural information from the data graphs. This section involves two key components: first, the automorphism orbits algorithm, which aids in organising subgraphs into groups of "similar" nodes for subsequent isomorphism counting. Second, the isomorphism counting itself, where we employ the orbits to tally the occurrences of isomorphic structures within the data graph, utilising the edges as anchor points. These two algorithms collectively furnish us with structural information  $\mathbf{x}^E$  that aligns with the list of subgraphs to be incorporated into the model.

In Section 4.3, titled "Subgraph DGSR Mechanism," we outline how the model manages graph embeddings and processes the extracted information to make predictions. As depicted in Fig. 1, our model comprises  $l$  DGSR layers, interconnected via our subgraph message passing scheme. Each DGSR layer begins with long-term and short-term embedding processes for both users and items, detailed in Section 4.3.1. Subsequently, a message passing module, elaborated upon in Section 4.3.2, combines these embeddings with structural information to carry out its function. In Section 4.3.3, we elucidate how, after processing

through these  $l$  layers, user embeddings are employed to generate sequential predictions. In Section 4.3.4, we provide insights into how we derive importance scores for the subgraphs from a trained subgraph DGSR model.



Fig. 1: The overview of Subgraph DGSR framework, taking the prediction of  $u_1$  with interaction sequence  $(i_1, i_2, i_3)$  as an example. Starting at the bottom left corner, we convert users' interaction sequences into dynamic graphs  $G_{t_3}$ , and perform Algorithm 1 and get a  $m$ -order sample graph  $G_{u_1}^m(t_3)$  (Section 4.1). Before imputed to the model, we perform Algorithm 3 (Section 4.2.2) to generate structural information from  $G_{u_1}^m(t_3)$  with orbits of the subgraph list. The orbits are counted by Algorithm2 (Section 4.2.1) when the model is set up. After this, the well-designed Subgraph DGSR model (Algorithm 4; Section 4.3) will propagate and aggregate the information and finally output the prediction

Looking at the general structure, the Isomorphism Counting part and DGSR part collaborate and focus on distinct aspects: the Isomorphism Counting algorithm embeds structural information, abstracted from specific users and items, and forwards it to the DGSR layers; while the embeddings of long-term and short-term user-item interactions are formulated in DGSR to pinpoint characteristic user preferences or item attributes, and are then propagated to the subsequent layer. Each time the structural information from Isomorphism Counting is integrated with the embeddings from DGSR and updated to the next layer, the model's structural memory is enhanced. A simplified depiction of the synergy between Isomorphism Counting and DGSR is presented in Fig. 2. In general, Isomorphism Counting works on the

structures of user-item interactions regardless of specific users or items, and then passed on to DGSR, which focuses on embedding specific user and item information.

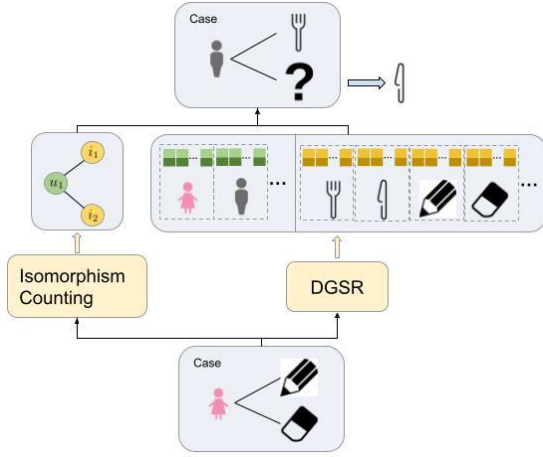


Fig. 2: The role of Isomorphism Counting and DGSR in our model. Isomorphism Counting extracts the structure information and DGSR process embedding on users and items. Both of them collaborate to generate predictions.

## 4.1 Graph Generation

In this section, we present the process of transforming user behaviour record data into graph structures, which primarily involves two key steps: Dynamic Graph Construction and Subgraph Sampling, as previously established in the DGSR model by Zhang (2022).

### 4.1.1 Dynamic Graph Construction

In this section, we outline the method to transform all user sequences into a dynamic graph. When a user  $u$ , interacts with an item  $i$ , at a specific timestamp  $t$ , we establish an edge  $e$ , between  $u$  and  $i$ . This edge can be characterised by a quintuple  $(u, i, t, o_u^i, o_i^u)$ , where  $t$  denotes the timestamp when the interaction took place;  $o_u^i$  represents the order of interaction between  $u$  and  $i$ , essentially signifying the item  $i$ 's position among  $u$ 's interactions;  $o_i^u$  denotes the order of user  $u$  among all the user nodes that have interacted with item  $i$ .

After that, we forms a dynamic graph denoted as  $G = \{(u, i, t, o_u^i, o_i^u) | u \in U, i \in V\}$ , which records the interaction times between users and items and captures the order information between them. Subsequently, we define our dynamic graph at a specific time,  $t$ , as  $G^t \in G$ , which comprises all users' interaction sequences up to and including time  $t$ . For a

given user sequence,  $S^u = (i_1, i_2, \dots, i_k)$ , with the corresponding timestamp sequence  $T^u = (t_1, t_2, \dots, t_k)$ , predicting the next item in sequence  $S^u$  is equivalent to predicting the item linked to the node  $u$  in the dynamic graph at time  $t_k$ , denoted as  $G^{t_k}$ .

#### 4.1.2 Subgraph Sampling

As the user sequence  $S_u$  expands, the number of neighbour sequences grows exponentially, which can be computationally costly and introduce substantial noise into our target sequence  $S_u$ . To manage computational complexity and reduce noise in user sequences, we propose an efficient sampling strategy. We begin by selecting a user node  $u$  as the anchor node and extracting its most recent  $n$  first-order neighbours from graph  $G^{t_k}$ , representing the historical items  $u$  has interacted with. Then, for each item  $i \in N_u$ , we use it as an anchor node to sample the set of users who have interacted with that item, denoted as  $N_i$ . To enhance sampling efficiency, we keep track of nodes used as anchor nodes to prevent redundant sampling and limit the maximum number of samples to  $n$  when sampling user nodes. This process allows us to obtain the multi-hop neighbours of node  $u$ , forming the user  $u$ 's  $m$ -order subgraph  $G_u^m(t_k)$  of  $S^u$ , where  $m$  is a hyper-parameter controlling the sub-graph's size. A pseudo algorithm is shown below:

---

#### Algorithm 1: Graph Sampling

---

**Input:** User sequence  $S^u = (i_1, i_2, \dots, i_k)$ ;  
Time sequence  $T^u = (t_1, t_2, \dots, t_k)$ ;  
Dynamic graph  $G^{t_k}$ ;  
Order of graph  $m$ ;

**Output:** The  $m$ -order Graph  $G_u^m(t_k)$ .

// Initialization

$U_m, U_{temp} \leftarrow \{u\}, I_m, I_{temp} \leftarrow \emptyset$

// Node Sampling

**for each**  $k \in [1, \dots, m]$ :

**if**  $k$  is an odd number:

**for each**  $u \in U_{temp}$ :

$I_{temp} \leftarrow I_{temp} \cup N_u$

$I_{temp} \leftarrow I_{temp} \setminus I_m$

**if**  $I_{temp} = \emptyset$ :

        Break

---

---

```

 $I_m \leftarrow I_m \cup I_{temp}$ 
else:
  for  $i \in I_{temp}$ :
     $U_{temp} \leftarrow U_{temp} \cup N_i$ 
   $U_{temp} \leftarrow U_{temp} \setminus U_m$ 
  if  $U_{temp} = \emptyset$  :
    Break
  if  $\|U_{temp}\| > n$  :
    Sample  $n$  nodes from  $U_{temp}$ 
     $U_m \leftarrow U_m \cup U_{temp}$ 
//Graph Generation:
 $G_u^m(t_k) = (U_m, I_m)$ , where  $U_m, I_m \in G^{t_k}$ 

```

---

## 4.2 Isomorphism Counting

We first illustrate how the model performs isomorphism counting in an intuitive way (see Fig. 3). According to Bouritsas's message passing scheme (2022), isomorphism counting involves node or edge counts, with edges capturing richer information by connecting two nodes. In our context, an edge represents a user's behaviour, which is more contextually meaningful in recommender systems compared to individual user or item nodes. Therefore, we prioritise the isomorphism counting by edges in our methodology.

When we get a list of subgraphs, we need to categorise the edges of a subgraph into distinct sets (the orbits) which act similarly within an orbit. This step is done by *automorphism orbit counting* (section 4.2.1), where each node is matched to other isomorphic nodes within the same graph (automorphism). As a result, we get subgraphs with orbit partitions as shown in Fig. 3 with different colours (both subgraph 1 and 2 have only one edge orbit because they are symmetry). Subsequently, we employ this partitioning of orbits to perform *isomorphism counting* (section 4.2.2) within the data graph for each unique orbit. We take each edge in the data graph in turns as the anchor edge (depicted as the red edge), to count the conditions that the subgraph is contained in the data graph with the anchor node taking the position of the edge of the orbit. It's notable that, within an orbit distinct edges are treated as equivalent, so in Fig. 3 the red edge can take the position of each of the blue edges (or yellow edges for subgraph 2), because they are symmetry (i.e. in an orbit). Consequently, the counts of all orbits related to the anchor node are appended to its features as structural information.



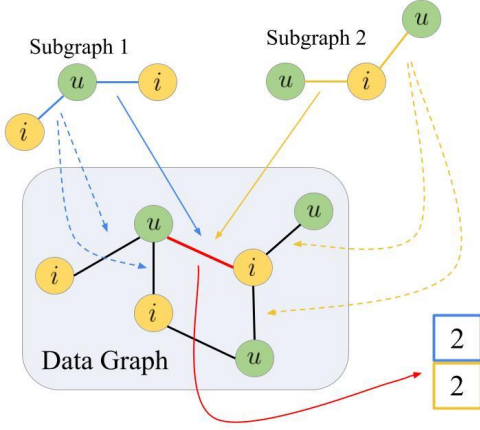


Fig. 3. Isomorphism counting on Data Graph from a list of 2 subgraphs. The red edge in the Data Graph is the anchor edge that we count for. Blue and yellow refer to orbits. Dash arrows represent 2 options when matching, so counted as 2. The boxes at bottom right corner are results attached to the anchor edge.

We have illustrated how the model actually performs the isomorphism counting on data graphs with a list of subgraphs, which yields isomorphism counts by orbits attached to the graph as structural features. In the following part of this section, we mainly focus on how we represent the chosen subgraph list, which serves as a set of hyperparameters for our model, covering two essential steps: *automorphism orbit counting* for the selected subgraphs and *isomorphism counting* by edges, as we mentioned above.

At the start of the model, we select the list of  $k$  subgraphs that require the model's attention, denoted as  $\mathcal{H} = \{H_1, H_2, \dots, H_k\}$ , where each element  $H_i$  represents an individual subgraph from our list. Subsequently, we will compute the automorphism orbit for each of these selected subgraphs.

#### 4.2.1 Automorphism Orbit Counting

This algorithm introduces how we discern automorphisms in a graph using its edge list. (Automorphisms are symmetries allowing graph parts to be interchanged without changing the graph's overall structure.)

Starting with an given edge list  $E_{H_i}$ , we construct a NetworkX graph  $H_i(V_{H_i}, E_{H_i})$ . (In our scripts, the input graphs are bi-directed with edges from both “users to items” and “items to users” as well. Then, considering that self-loops in a graph can overshadow information from neighbouring nodes, we first need to refine  $E_{H_i}$  by excluding self-loops:  $E_{H_i} = E_{H_i} \setminus \{(v, v) | v \in V_{H_i}\}$ . From the cleaned graph, we extract its automorphisms as

$\mathcal{A}_i = \text{Automorphisms}(H_i)$ , where we use NetworkX's GraphMatcher to detect automorphisms. In  $H_i$ , each vertex starts with an initial orbit, expressed as  $\mathcal{M}(v) = v, \forall v \in V_{H_i}$ , before updating the orbits into this collection. However, in found automorphisms, the vertices in an orbit act as the same ‘‘role’’ and we need one certain vertex to represent this orbit, so we update each vertex's orbit to its minimal label using:  $\mathcal{M}(v) = \min(\mathcal{M}(v), a(v)), \forall v \in V_{H_i}, \forall a \in \mathcal{A}_i$ . By using the minimum label, we consistently pick one certain vertex as the label of their shared orbit.

Once orbits are finalised, vertices are clustered by their orbit classifications:  $\mathcal{P} = \{v | \mathcal{M}(v) = o\}, \forall o \in \text{unique}(\mathcal{M})$ . The entire process concludes by counting the discovered automorphisms, represented as  $\mathcal{A}_i = \text{length}(\mathcal{A}_i)$ . And then we pack the outputs including the refined graph, orbit partitions, an orbit membership dictionary, and a count of detected automorphisms.

The detailed pseudocode is provided below:

---

**Algorithm 2: Automorphism Orbit Counting**

---

**Input:** edge list of a subgraph:  $E_{H_i} = \{(a_1, b_1), \dots, (a_{s(i)}, b_{s(i)})\}$ ,  
where  $a_p, b_p \in V_{H_i}, s(i) = \|E_{H_i}\|$

**Output:** A quadruple  $(H_i, \{\mathcal{O}_{H_i}^E(a, b) | (a, b) \in E_{H_i}\}, \mathcal{M}_i, |\mathcal{A}_i|)$ ,  
where  $\mathcal{O}_{H_i}^E(a, b)$  represents orbits of edge  $(a, b)$ ,  
 $\mathcal{M}_i$  indicates the orbit membership dictionary of  $H_i$ ,  
and  $|\mathcal{A}_i|$  refers to the count of automorphisms of  $H_i$ .

// construct subgraph

$H_i \leftarrow E_{H_i}$ ,

// remove self-loops

$H_i \leftarrow H_i \setminus \{(v, v) | v \in V_{H_i}\}$

// find automorphisms

$\mathcal{A}_i = \emptyset$

**for**  $(v_1, v_2) \in V \times V$ :

    initialise  $f : V \rightarrow V$ , where  $f(v_1) = v_2$

**for**  $a \in N(v_1)$ :

        search  $w \in N(v_2)$  within  $f$

        add  $f(a) = w$  to  $f$

    // validate if it's an automorphism

---

---

```

is_automorphism = True
for  $(a, b) \in E_{H_i}$ :
    if  $(f(a), f(b)) \notin E_{H_i}$ :
        is_automorphism = False;
for  $(f(a), f(b)) \in E_{H_i}$ :
    if  $(a, b) \notin E_{H_i}$ :
        is_automorphism = False;
if is_automorphism:
     $\mathcal{A}_i^+ = f$ 
for  $a \in \mathcal{A}_i$ :
    for  $e \in E_{H_i}$ :
         $\mathcal{M} \leftarrow \mathcal{O}_{H_i}^E(a, b)$ 
\\group vertices by  $\mathcal{O}_{H_i}^E(a, b)$ 
 $\mathcal{P} = \{\}$ 
for  $e \in E_{H_i}$ :
     $o = \mathcal{M}(e)$ 
    if  $o \notin \mathcal{P}$ :
         $\mathcal{P}[o] = v$ 
 $|\mathcal{A}_i| = \text{len}(\mathcal{P})$ 

```

---

#### 4.2.2 Subgraph Isomorphism Counting

Once we have computed the orbits for the subgraphs, we can proceed to calculate the structural features, following the method outlined in Bouritas' research: Consider the list of subgraphs  $\mathcal{H} = \{H_1, H_2, \dots, H_k\}$ , for each graph  $H_i \in \mathcal{H}$ , we initially identify its isomorphic subgraphs in the  $G$ , represented as  $G_S$ . Then, for each edge  $(a, b) \in E_{G_S}$ , we determine its significance concerning a particular graph by examining the orbit  $\mathcal{O}_{H,i}^E(a, b)$  of its corresponding mapping  $(f(a), f(b))$  in  $H_i$ . By counting the occurrences of various orbits of  $(a, b)$ , we derive the vertex structural feature  $\mathbf{x}_{H_i}^E(a, b)$  of edge  $(a, b)$ , defined as follows. For all  $j \in \{1, \dots, d_H\}$ :

$$x_{H_i,j}^E(a, b) = |\{G_S \cong H_i \mid (a, b) \in E_{G_S}, (f(a), f(b)) \in \mathcal{O}_{H_i,j}^E\}|,$$

where we count for the anchor edge  $(a, b)$  with the orbit  $j$  in subgraph  $H_i$ ,

$$\text{and } \mathbf{x}_{H_i}^E(a, b) = [x_{H_i,1}^E(a, b), \dots, x_{H_i,d_H}^E(a, b)],$$

where we aggregate the counts crossing different orbits within subgraph  $H_i$ ,

and thus combine the edge features  $\mathbf{x}_{(a,b)}^E = [\mathbf{x}_{H_1}^E(a,b), \dots, \mathbf{x}_{H_k}^E(a,b)]$  as the structural features. The detailed pseudocode is provided below:

---

**Algorithm 3: Isomorphism Counting**

---

**Input:** Graph  $G_u^m(t_k)$ , denoted as  $G$  below,  
List of subgraphs  $\mathcal{H} = \{H_1, H_2, \dots, H_k\}$ .

**Output:** Structure features  $\mathbf{x}_{(a,b)}^E$ .

// Remove self-loops  
 $G \leftarrow G \setminus \{(v,v) | v \in V_G\}$

// Compute the isomorphic counts

**for**  $H_i \in \mathcal{H}$ :

    // Use GraphMatcher from Networkx (VF2 algorithm)  
isomorphism maps  $\mathcal{I} \leftarrow \{\{(a, f(a)) | a \in H_i, f(a) \in G\} | f \in V \times V\}$ ,  
    where  $V \times V$  denotes all isomorphism functions.

    // Initialise the structure features for  $H_i$   
 $x_{H_i,j}^E(a,b) = 0$   
 $\mathbf{x}_{H_i}^E(a,b) = [x_{H_i,1}^E(a,b), \dots, x_{H_i,d_H}^E(a,b)]$

    // Count by isomorphism maps

**for**  $iso \in \mathcal{I}$ :

**for**  $(a,b) \in E_{H_i}$ :

**if**  $(a, f(a)), (b, f(b)) \in iso$ :

$count+ = 1$

    // Normalise the counts  
 $x_{H_i,j}^E(a,b) \leftarrow count \setminus |\mathcal{A}_i|$

    // Concatenate structure features  
 $\mathbf{x}_{(a,b)}^E = [\mathbf{x}_{H_1}^E(a,b), \dots, \mathbf{x}_{H_k}^E(a,b)]$

---

### 4.3 Subgraph DGSR Mechanism

In this section, we provide an overview of the fundamental structure of our Subgraph DGSR model, which is divided into four key sections: In section 4.3.1, we delve into the rationale and methods behind executing long-term and short-term embeddings for users and items within our Subgraph DGSR layer. Moving on to section 4.3.2, we explain our approach to

aggregating and concatenating information, updating new user and item features for propagation to the subsequent layer. After progressing through multiple layers, section 4.3.3 outlines how we generate personalised recommendations for users. Lastly, in section 4.3.4, we detail the process of generating importance scores using the trained model.

### 4.3.1 Information Encoding

In recommender systems, the conversion of user-item records into bipartite graphs and the division of message passing into two parts, item-user and user-item interactions, stem from two key considerations. 1. interactions are often asymmetric, with users engaging with items, but not the same way back; 2. users and items contain distinct data types; users have demographic and behavioural information, while items possess unique product features. In our model, we design another partition for embedding: long-term (L) and short-term (S), where long-term refers to the user's preference in the history of interactions, and short-term for the most recent purchase (or behaviour) they make. Overall, we have 4 embedding paths in our model: 1. Long-term user-to-item; 2. Short-term user-to-item; 3. Long-term item-to-user; 4. Short-term item-to-user. For brevity, “user-to-item” is depicted as  $ui$  in the following equations and vice versa.

DGRN combines the strengths of the Graph Attention mechanism and sequence encoding, and designs a dynamic attention module to capture the interaction order information, due to the following reasons: traditional static GNNs like GCN and GAT excel in processing various graph structure data, but they often overlook the sequential information of neighbours for each user and item; while sequence-based models like RNN and Transformer nets need to be invoked for more dynamic scenarios. By this combination, the attention coefficients are then influenced by a relative-order-aware attention mechanism, making it possible to compute long-term preferences of users and items:

$$\mathbf{h}_u^L = \sum_{i \in \mathcal{N}_u} \alpha_{ui} \left( \mathbf{W}_1^{(l-1)} \mathbf{h}_i^{(l-1)} + \mathbf{p}_{r_u^V} \right),$$

$$\mathbf{h}_i^L = \sum_{u \in \mathcal{N}_i} \beta_{iu} \left( \mathbf{W}_2^{(l-1)} \mathbf{h}_u^{(l-1)} + \mathbf{p}_{r_i^U} \right),$$

where  $\alpha_{ui} = \text{softmax}(e_{ui})$ ,  $\beta_{iu} = \text{softmax}(e_{iu})$ ,

$$\text{and } e_{ui} = \left( \mathbf{W}_2^{(l-1)} \mathbf{h}_i^{(l-1)} \right)^T \left( \mathbf{W}_1^{(l-1)} \mathbf{h}_u^{(l-1)} + \mathbf{p}_{r_u^V} \right) \setminus \sqrt{d},$$

$$e_{iu} = (\mathbf{W}_1^{(l-1)} \mathbf{h}_u^{(l-1)})^T (\mathbf{W}_2^{(l-1)} \mathbf{h}_u^{(l-1)} + \mathbf{p}_{r_i^u}^V) \setminus \sqrt{d},$$

and  $\mathbf{P}_{r_i^u}^V$  refer to order-relative embedding.

Complementing the long-term information, the short-term information focuses on capturing users' recent interests. Unlike traditional approaches that emphasise only the last interaction, DGRN leverages an attention mechanism to balance the latest interaction with historical ones:

$$\mathbf{h}_u^S = \sum_{i \in \mathcal{N}_u} \hat{\alpha}_{ui} \mathbf{h}_i^{(l-1)},$$

$$\mathbf{h}_i^S = \sum_{u \in \mathcal{N}_i} \hat{\beta}_{iu} \mathbf{h}_u^{(l-1)},$$

where  $\hat{\alpha}_{ui} = \text{softmax}(\hat{e}_{ui})$ ,  $\hat{\beta}_{iu} = \text{softmax}(\hat{e}_{iu})$ ,

$$\text{and } \hat{e}_{ui} = (\mathbf{W}_2^{(l-1)} \mathbf{h}_{i_{last}}^{(l-1)})^T (\mathbf{W}_1^{(l-1)} \mathbf{h}_i^{(l-1)} + \mathbf{p}_{r_i^u}^V) \setminus \sqrt{d},$$

$$\hat{e}_{iu} = (\mathbf{W}_1^{(l-1)} \mathbf{h}_{u_{last}}^{(l-1)})^T (\mathbf{W}_2^{(l-1)} \mathbf{h}_u^{(l-1)} + \mathbf{p}_{r_i^u}^V) \setminus \sqrt{d}.$$

### 4.3.2 Node Updating

With the propagation information well calculated, the node features need to be updated. This process enables nodes to concatenate information from long-term and short-term interactions, as well as from the preceding layer, resulting in a new comprehensive representation for users and items. Moreover, we will incorporate the edge path from the Graph Substructure Network (GSN - e) introduced by Bourtias (2022), integrating the isomorphism counting results from Section 4.2 into the passing message, as outlined in the following scheme:

(From now,  $\mathbf{x}_{(a,b)}^E$  is denoted as  $\mathbf{x}_{(u,i)}^E$  in the context of recommendation systems.)

$$\mathbf{h}_u^{(l)} = \tanh(\mathbf{W}_5^{(l)} \times [\mathbf{h}_u^L \parallel \mathbf{h}_u^S \parallel \mathbf{h}_u^{(l-1)} \parallel \mathbf{x}_{(u,i)}^E])$$

$$\mathbf{h}_i^{(l)} = \tanh(\mathbf{W}_6^{(l)} \times [\mathbf{h}_i^L \parallel \mathbf{h}_i^S \parallel \mathbf{h}_i^{(l-1)} \parallel \mathbf{x}_{(i,u)}^E])$$

where  $\mathbf{W}_5^{(l)}, \mathbf{W}_6^{(l)} \in \mathbb{R}^{d \times 3d}$ , leveraging the information from long term, short term, embedding before updates and isomorphism counting.

### 4.3.3 Generate the Recommendations

From the described mechanism, following  $L$  layers of model propagation, we acquire numerous embeddings for each user node  $u$ . These embeddings are then concatenated as follows:

$$\mathbf{h}_u = \text{CONCAT}(\mathbf{h}_u^{(0)} \parallel \mathbf{h}_u^{(1)}, \dots, \parallel \mathbf{h}_u^{(L)}).$$

For a candidate item  $i$ , the link function is  $s_{ui} = \mathbf{h}_u^T \mathbf{W}_P \mathbf{e}_i$ , where  $s_u$  represents the user's scores for all candidate items. We learn the model parameters by optimising the cross-entropy loss, including regularisation:

$$\text{Loss} = - \sum_S \sum_{i=1}^{|\mathcal{I}|} (y_{ui} \log(\hat{y}_{ui}) + (1 - y_{ui}) \log(1 - \hat{y}_{ui})) + \lambda \|\Theta\|^2$$

where  $y_u$  encodes the actual items in  $S^u$  for the user  $u$ ,  $\Theta$  comprises all model parameters, and  $\lambda$  regulates the regularisation strength.

The detailed pseudocode algorithm of our entire model is provided below:

---

#### Algorithm 4: Subgraph DGSR Model Structure

---

**Input:**  $S^u = (i1, i2, \dots, ik)$ : Current sequence for user  $u$ ;  
 $T^u = (t1, t2, \dots, tk)$ : Timestamp sequence for user  $u$ ;  
 $L$ : DGRN layer number

**Output:**  $i_{k+1}$ : The next item of  $S^u$

// Generate graph data for  $S^u$

Run **Algorithm 1: Graph Sampling** to generate  $G_u^m(t_k)$  from  $G^{t_k}$

// Count automorphism orbits for subgraphs

Run **Algorithm 2: Automorphism Orbit Counting** to generate  $\mathbf{O}_{H_i}^V$

// Initialise node representation

**for**  $\forall u, i \in G_u^m(t_k)$ :

$\mathbf{h}_u^{(0)} = \mathbf{e}_u$

$\mathbf{h}_i^{(0)} = \mathbf{e}_i$

//Generate Isomorphism for messaging passing

Run **Algorithm 3: Isomorphism Counting** to generate  $\mathbf{x}_{(u,i)}^E$

// Update users and items

**for**  $l \in [1 : L]$ :

---

---


$$\mathbf{h}_u^{(l)}, \mathbf{h}_i^{(l)} = DGRN(\mathbf{h}_u^{(l-1)}, \mathbf{h}_i^{(l-1)}, G_u^m(t_k))$$

$$\mathbf{h}_u^L, \mathbf{h}_i^L \leftarrow LongTermEncoding$$

$$\mathbf{h}_u^S, \mathbf{h}_i^S \leftarrow ShortTermEncoding$$

$$\mathbf{h}_u^{(l)} \leftarrow \tanh(\mathbf{W}_5^{(l)} \times [\mathbf{h}_u^L \parallel \mathbf{h}_u^S \parallel \mathbf{h}_u^{(l-1)} \parallel \mathbf{x}_{(u,i)}^E])$$

$$\mathbf{h}_i^{(l)} \leftarrow \tanh(\mathbf{W}_6^{(l)} \times [\mathbf{h}_i^L \parallel \mathbf{h}_i^S \parallel \mathbf{h}_i^{(l-1)} \parallel \mathbf{x}_{(i,u)}^E])$$

// Predict the next item

$$\mathbf{h}_u = CONCAT(\mathbf{h}_u^{(0)} \parallel \mathbf{h}_u^{(1)}, \dots, \parallel \mathbf{h}_u^{(L)})$$

$$NextItem = \underset{i \in V}{\operatorname{argmax}} (\mathbf{h}_u^T \mathbf{W}_P \mathbf{e}_i)$$


---

#### 4.3.4 Importance Scores

From the subgraph isomorphism counting algorithm, we derive structural features from all graphs by mapping each graph to every subgraph. To get how these subgraphs are obtained in the data graph, we extract the isomorphism counts and aggregate them based on subgraphs (rather than the data graph), and then proceed with normalisation. Here are the specific steps outlining how we compute the importance scores for each subgraph.

For all  $j \in \{1, \dots, d_H\}$ :

$$\text{orbit importance } Imp_{H_i, j} = AGG_{G_u^m(t_k) \in \mathcal{G}}(x_{H_i, j}^E(a, b)),$$

$$\text{and then subgraph importance } Imp_{H_i} = Norm\left(\sum_{j=1}^{d_H} (Imp_{H_i, j})\right)$$

$$\text{where } x_{H_i, j}^E(a, b) = |\{G_u^m(t_k)_S \cong H_i \mid (a, b) \in E_{G_u^m(t_k)_S}, (f(a), f(b)) \in \mathcal{O}_{H_i, j}^E\}|,$$

$AGG$  indicates the aggregate function, and  $Norm$  represents the normalisation function. Subsequently, we obtain importance scores that represent the relative significance of different subgraphs in the list.

## 5. Experiments

In this section, we conduct a compelling experiment on Amazon user review data, seeking to comprehensively evaluate the performance of our proposed model compared to the original model, and introduce the innovative explainable metric framework. Throughout the entire experiment, we mainly address the following 3 research questions:



1. How does our proposed Subgraph DGSR perform compared to the original DGSR model?
2. What’s the effect of hyperparameters on the performance of our DGSR?
3. What factors of our input subgraphs influence the performance of DGSRs?

For RQ1, we test our Subgraph DGSR model in comparison to the original DGSR model across various datasets as depicted in section 5.1, under the same hyperparameter settings and evaluation methods detailed in section 5.2, and we display the performance of both models to showcase the superiority of our model; For RQ2, we test 3 hyperparameters: layer numbers, max. length of sequence, and sampling size to show the impact of changing them and give insights to further researchers. We change one of the model settings, retain the others as in RQ1, and compare the performance to test the impact of the changed hyperparameter. For RQ3, we mainly test on the list of subgraphs based on 3 perspectives: the number of subgraphs, the size of subgraphs and the user/item ratio. We design different lists of subgraphs for comparison, and we assess their influence on the models’ performance and the importance scores assigned to the subgraphs.

## 5.1 Dataset

In our study, we employ the Amazon Review Data (Jianmo Ni, Jiacheng Li, Julian McAuley, Empirical Methods in Natural Language Processing, EMNLP, 2019) obtained from the Stanford Network Analysis Project.

The dataset encompasses an extensive collection of over 100 million reviews, including essential details such as the review time, reviewers' identities, review contents, and other pertinent information. The dataset encompasses a diverse array of Amazon products spanning 29 distinct categories, with users and products identified through anonymized codes to preserve confidentiality. We choose 3 datasets: CD, Music and Grocery, for implementing our model as well as for the subsequent case study.

Given our primary emphasis on the data's structure, and for the sake of simplicity, we consider the subset of source data to be composed of (user\_id, product\_id, time) pairs, as depicted in Table 1. It is crucial to note that our model possesses the capability to accommodate a wide spectrum of features of real user behaviours (e.g. ratings, order information) that are encountered in real-world scenarios.

user_id	item_id	time
143026860	A1V6B6TNIC10QE	1424304000
014789302X	A1V6B6TNIC10QE	1491782400
014789302X	A2V9BG2MDQVCYX	1422748800

Table 1. Raw data from Amazon Review Data.

Analysing the data reveals that certain users exhibit behaviours with limited records across the entire dataset, thereby diminishing data availability. Consequently, we opted to adopt Zhang's data selection approach, which involves excluding users with fewer than 5 records. Following this filtration, we proceeded to randomly select records from the dataset and subsequently transformed the user\_id and item\_id into distinct integer representations, as shown in Table 2:

user_id	item_id	time
0	0	1424304000
1	0	1491782400
1	1	1422748800

Table 2. Processed data from Amazon Review Data.

We employ the previously mentioned node sampling technique to sample nodes and transform them into graphs. Since the sequences of interactions between users and items are unpredictable, resulting in varying graph sizes, the quantity of graphs differs across each dataset. The characteristics of each type of dataset (categories) are displayed in Table 3 :

Dataset	Interactions	User_ids	Item_ids	Train Graphs	Test Graphs
CD	10,000	7982	574	494	360
CD	20,000	14,018	1,101	2056	1043
Music	10,000	6914	1374	751	614
Music	20,000	11824	2391	2542	1654
Grocery	10,000	8821	380	65	137
Grocery	20,000	16748	607	266	453

Table 3. Size of each used dataset.

In addition, the average purchase length represents the mean length of users' purchase histories in the 3 datasets, excluding the invalid users with fewer than 5 orders, as shown in Table 4.

Dataset	AVE. length
CD	13.7
Music	11.0
Grocery	9.2

Table 4. Average purchase length of 3 datasets.

Given the practical nature of data, which is often provided as records rather than pre-defined graphs and can vary in size, we opt to categorise datasets based on recorded interactions before the conversion into graphs. In such a scenario, precisely controlling the number of train/test graphs becomes challenging.

## 5.2 Model Settings

### 5.2.1 Default Parameter Set-up

The diagram provided in Table 5 illustrates the fundamental parameter configurations we have established. These hyperparameters remain consistent throughout the entire experiment, unless when conducting experiments specifically focused on the particular hyperparameter.

Hyperparameters	Values	Hyperparameters	Values
Epoch	10	LR	0.001
Num. DGSR layers	3	L2	0.0001
Optimizer	Adam	Item max length	10
Batchsize	32 (graphs)	User max length	10
Sampling Size (K-hop)	3 steps		

Table 5. Model settings for both subgraph DGSR and original DGSR models.

**Epoch:** the training epoch. Both models are fully trained with 10 epochs;

**Num DGSR layer:** the number of stacked DGSR layers we use. According to Zhang's work, 3 DGSR layers have the best performance;

**Sampling size** indicates the nodes and edges up to a certain distance (k steps) away from a given node in order to capture local and global information for message passing and aggregation. When using the value 3, we approach message passing in a sequence such as: *item-user-item-user*.

**Batchsize:** To enhance efficiency during iterations, we employ a batch size of 32 graphs for batch learning;

**LR:** The learning rate is 0.001;

**L2:** the `weight_decay` parameter of the optimizer. Set to 0.0001 to prevent overfitting;

**User max length:** the max length of item sequence for each user when sampling nodes;

**Item max length:** the max length of user sequence for each item when sampling nodes;

## 5.2.2 Evaluation Methods

In assessing the performance of all methods, we employ two popular metrics for sequential recommendations, Hit@K and NDCG@K (Kang, W. C. & McAuley J., 2018; Li, J. et al., 2020). The Hit@K metric calculates the proportion of relevant items that are successfully retrieved among all relevant items, typically within the top K retrieved results. Conversely, NDCG@K is a position-sensitive measure, where a higher NDCG value implies that desired items are more likely to have higher ranks.

The calculator of Hit@K is:

$$Recall@K = \frac{Re_K}{Re_{All}}$$

where  $Re_K$  indicates the number of relevant items in top  $K$ , and  $Re_{All}$  for the total number of relevant items.

And the calculation for NDCG@K:

$$DCG@K = \sum_{i=1}^K \frac{2^{r_i} - 1}{\log_2(i + 1)}$$

$$IDCG@K = \sum_{i=1}^K \frac{1}{\log_2(i + 1)}$$

$$NDCG@K = \frac{DCG@K}{IDCG@K}$$

where  $r_i$  represents the relevant score of the item. DCG@K indicates discounted cumulative gain, measuring how well the algorithm's ranking captures the relevant information. And

IDCG@K indicates ideal discounted cumulative gain, as the baseline for the best possible ranking.

Following the approach of Kang (2018) and Li (2020), our evaluation is conducted on a per-test-sample basis. For each test sample, we randomly choose 100 items not included in the ground truth and rank them alongside the true item. This results in a set of 101 items per test sample, forming the basis for our Hit@K and NDCG@K assessments. We choose the values 5, 10, and 20 for K, resulting in Hit@5, Hit@10, Hit@20, as well as NDCG@5, NDCG@10, and NDCG@20 metrics for model comparison.

### 5.2.3 Environment & Parameter Setup

The computing environment is Google Colaboratory. For DGSR model building, the following packages are used: NumPy, Pandas, Torch and DGL; and for subgraph isomorphism metrics, NumPy, DGL, Torch\_Geometric and Networkx are used.

Bouritsas (2022) uses the Graph-tool package for graph computation and counting isomorphism and automorphism. However, due to compatibility issues with DGL in Colab, we rebuilt the subgraph isomorphism function using DGL and Networkx. This reconstruction follows the VF2 algorithm (Cordella et al., 2001; 2004), which is employed by Graph-tool (Tiago P. Peixoto, 2014).

### 5.2.4 Subgraph List

For our proposed Subgraph DGSR, we need to pre-define a set of subgraphs as hyperparameters that the model should take into special consideration. Our model can integrate subgraphs of any numbers and structures, but the input subgraphs should be designed as scenarios aligned with real-world purchase conditions and worth valuable research insights. In section 5, we design 3 basic subgraphs that represent 3 basic interactions that happen between users and items, as illustrated in the graphs in Table 6, and explained in the following paragraph.

(For further guidelines of subgraphs, we will provide the rules of creating a subgraph list in RQ3 and illustrate the framework of utilising subgraphs with our model in the section 6, Case Study.)

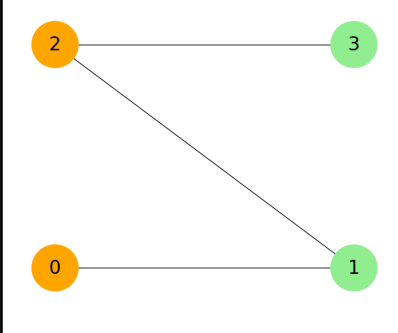
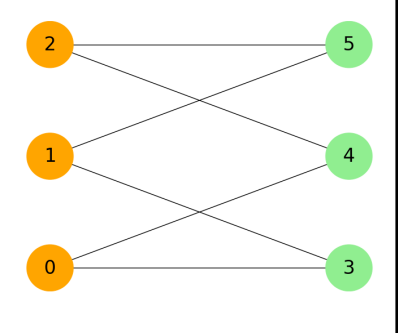
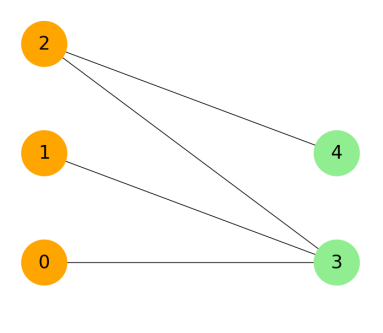
Recommends	Multi-interests	Recommends2
		

Table 6: The list of subgraphs used in RQ1 and RQ2. User nodes are shown in orange and item nodes in green.

**Recommends:** User-0 and user-2 have the same purchase of item-1, while user-2 also buys item-3. This scenario indicates one basic logic of recommendations “user-user collaborative filtering”, which indicates “people like you, like that” (e.g. Resnick, P. et al., 1994);

**Multi-interests:** Among the three users, each pair of them shares preference for the same product. This scenario refers to one of the common limitations of traditional collaborative filtering recommender systems, arising from the fact that users often have multiple interests, but collaborative filtering systems struggle to provide accurate recommendations because the recommended item for an active user may not align with the common interests of their neighbouring (similar) users or in a group of similar users (Koren, Y., 2008; Li, Y. et al., 2005; Strub, F. et al., 2016). To be aware of the multi-interests of users, we include a “multi-interests” graph of 3 users;

**Recommends2:** User-0,1,2 share the same interest as item-3, but user-2 has additional preference to item-4. This scenario serves as the complement of Recommends to better identify item-3: Recommends leads a too strong indication of similarity and might mislead the model when item-3 is not of interest to user-0, while Recommends2 helps in recognizing item-3 as an interest by involving an additional user. This helps filter out cases when 2 users coincidentally buy the same item.

During our investigation of RQ1 and RQ2, we incorporate these three predefined subgraphs into our model as the default list to ensure that our model maintains awareness of them. However, the comprehensive impact of input subgraphs on our model's performance will be fully explored in the section of RQ3, where we input more different subgraphs to test.

## RQ 1 Performance Comparison

In this section, we focus on addressing the first research question: comparing the performance of our novel Subgraph DGSR model with the original DGSR model, while also providing plausible explanations for the results obtained. The importance scores, which indicate the relative significance of subgraphs, are excluded from our model comparison and conclusions in RQ1. These scores are provided in Appendix 1 but are not a focal point in our analysis.

Datasets	Models	Hit@5	Hit@10	Hit@20	NDCG@5	NDCG@10	NDCG@20
CD 10,000	Subgraph DGSR	<b>0.325</b>	<b>0.442</b>	<b>0.597</b>	<b>0.235</b>	<b>0.272</b>	<b>0.310</b>
	Regular DGSR	0.264	0.383	0.539	0.173	0.214	0.253
CD 20,000	Subgraph DGSR	<b>0.409</b>	0.474	<b>0.692</b>	<b>0.365</b>	<b>0.386</b>	<b>0.429</b>
	Regular DGSR	0.385	<b>0.537</b>	0.657	0.257	0.304	0.343
Music 10,000	Subgraph DGSR	0.204	<b>0.350</b>	<b>0.533</b>	0.131	0.172	<b>0.234</b>
	Regular DGSR	<b>0.222</b>	0.308	0.457	<b>0.166</b>	<b>0.190</b>	0.228
Music 20,000	Subgraph DGSR	<b>0.652</b>	<b>0.741</b>	<b>0.845</b>	<b>0.584</b>	<b>0.611</b>	<b>0.637</b>
	Regular DGSR	0.598	0.649	0.720	0.540	0.554	0.575
Grocery 10,000	Subgraph DGSR	<b>0.409</b>	0.475	<b>0.577</b>	<b>0.363</b>	<b>0.375</b>	<b>0.394</b>
	Regular DGSR	0.380	<b>0.482</b>	0.547	0.334	0.361	0.393
Grocery 20,000	Subgraph DGSR	<b>0.567</b>	<b>0.634</b>	<b>0.742</b>	0.492	<b>0.526</b>	<b>0.551</b>
	Regular DGSR	0.545	0.614	0.717	<b>0.510</b>	0.515	0.539

Table 7. The performance of the models we use on 3 datasets with 2 different record partitions.

According to Table 7, our Subgraph DGSR achieves better performance on all 6 metrics within datasets of CD 10,000 and Music 20,000. On the other datasets, our model performs partially better than the original DGSR model. The worst condition is Music 10,000, where Hit@10, Hit@20 and NDCG@20 take better scores. A reasonable explanation for the

fluctuation of model performance can be, our input lists of subgraphs fail to match the inherent patterns in Grocery 20,000. On comparison of same datasets with different numbers of interactions, data size may also have an impact on performance, with larger datasets potentially presenting different challenges in recommendation.

Based on the performance of the CD dataset shown in Fig. 4, we have the following findings: 1. Subgraph DGSR consistently outperforms Regular DGSR across datasets of varying sizes. With the exception of the CD\_20000 dataset, Subgraph DGSR consistently exhibits superior performance compared to Regular DGSR. Moreover, this performance gap remains stable across various prediction list lengths, ranging from 5 to 20. This stability underscores the effectiveness of our proposed model in diving deeper into graph patterns compared to the regular model. 2. As the predicted list gets longer, the Hits and NDCGs are expected to get better within the same. The increase in Hit and NDCG scores with the expansion of the prediction list suggests that, even in scenarios where there are 100 negative items, predicting sequences of less than 20 items doesn't significantly impact the model's performance.

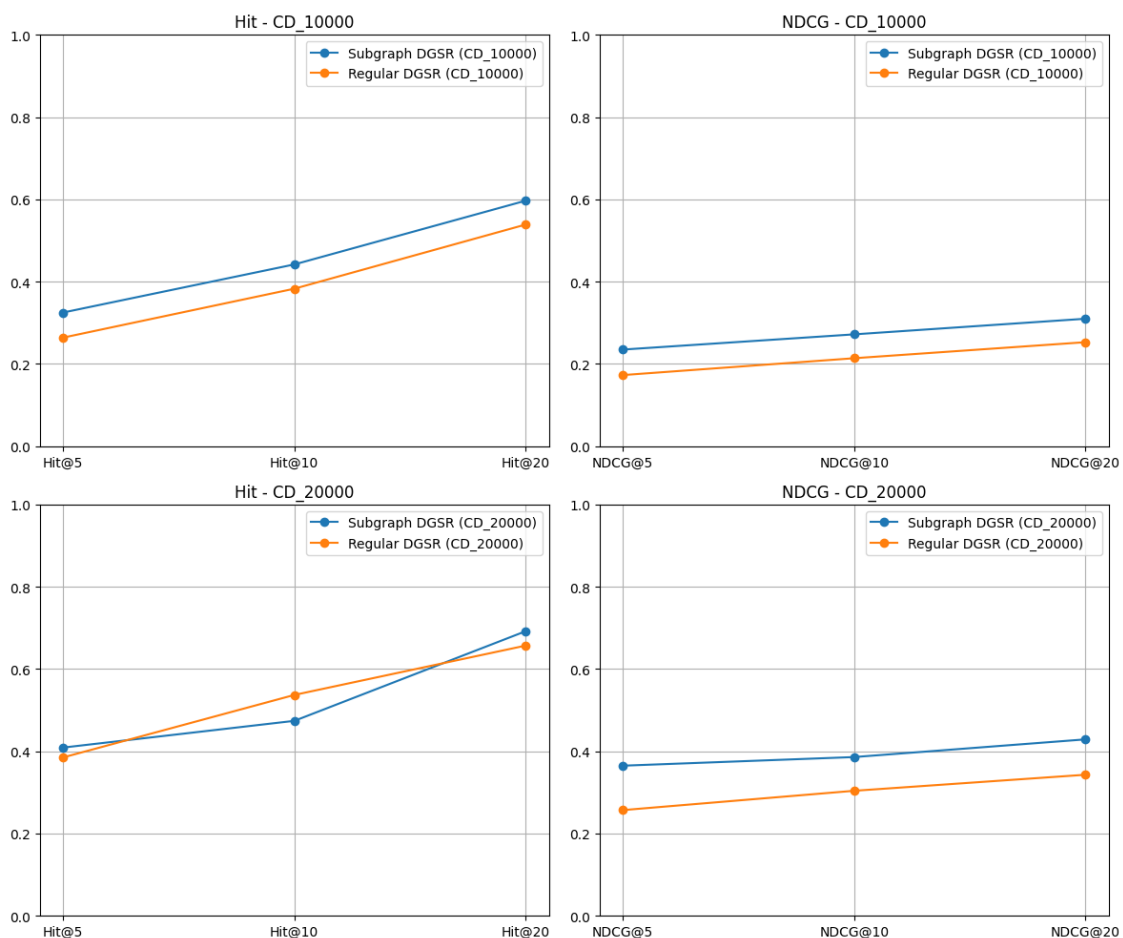




Fig. 4: The Hits (left) and NDCGs (right) performed on CD dataset with 10,000 (up) and 20,000 (down) interactions.

In the context of the Music dataset (Fig. 5), Music 20000 outperforms Music 10000 overall, which can be attributed to the availability of more training data. The Hit trendline on the left, comparing Subgraph DGSR and Regular DGSR, is no longer parallel; instead, it diverges as the predicted list length increases. Notably, our proposed model exhibits a faster growth rate, indicating its superior performance in predicting longer sequences. This also suggests that our default subgraphs effectively capture the patterns within the Music dataset. On the right side, in terms of NDCG, Music 10000 exhibits similar performance, while in Music 20000, our Subgraph DGSR maintains a moderate advantage over the regular model.

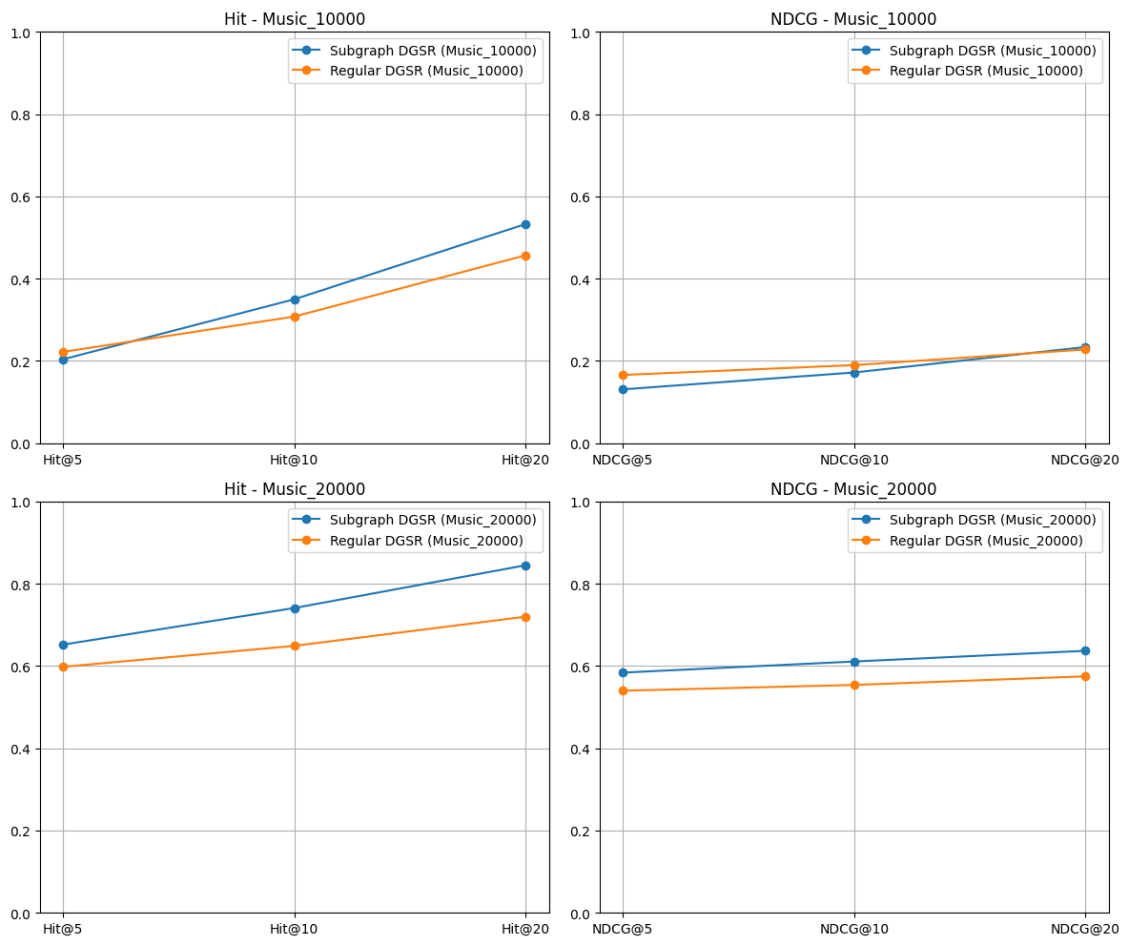


Fig. 5: The Hits (left) and NDCGs (right) performed on Music dataset with 10,000 (up) and 20,000 (down) interactions.

In the context of the Grocery dataset, as depicted in Fig. 6, the performance gap between the two datasets is relatively smaller when compared to certain other models. However, our Subgraph DGSR model manages to maintain a slight edge over the original DGSR model in

most of the evaluation metrics, regardless of the dataset's scale. This suggests that our model's superiority is consistent across different dataset sizes and reinforces its robust performance.

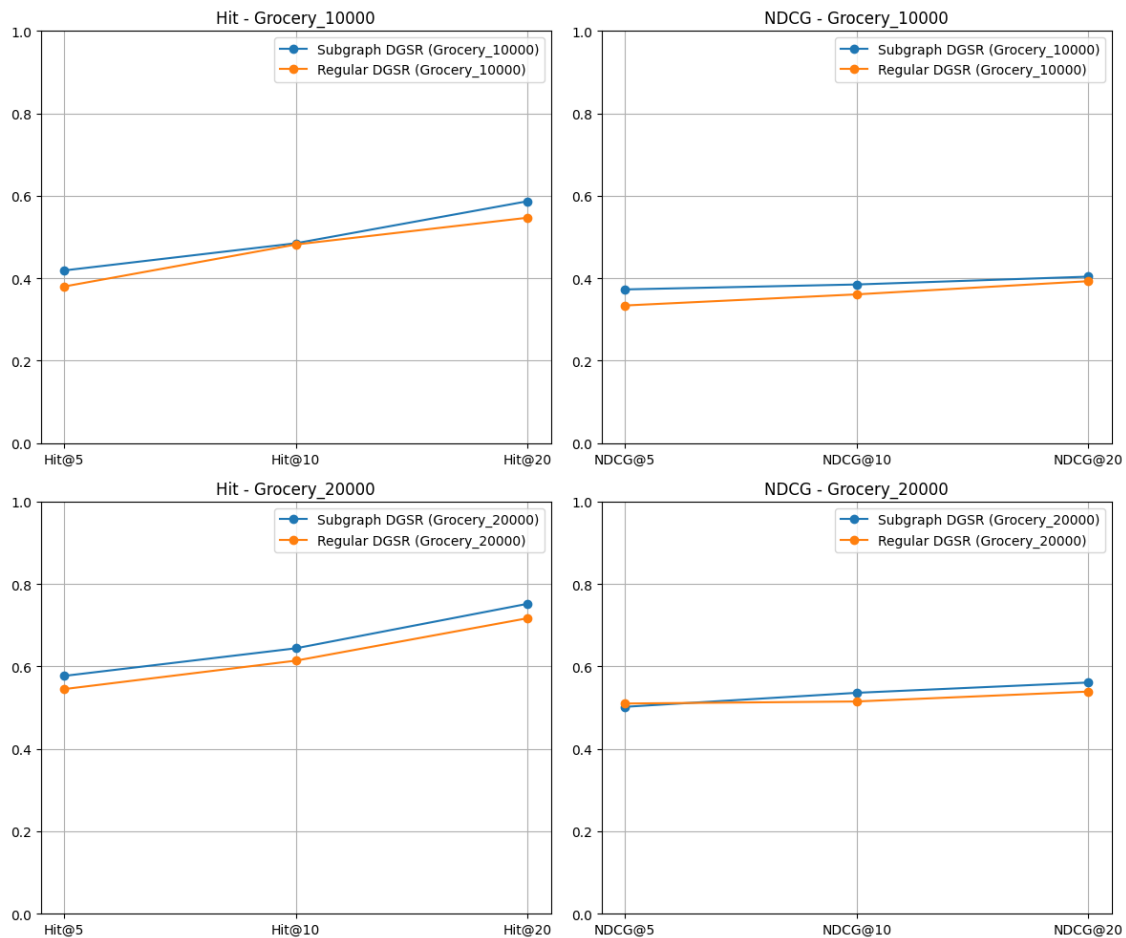


Fig. 6: The Hits (left) and NDCGs (right) performed on Grocery dataset with 10,000 (up) and 20,000 (down) interactions.

## RQ 2 Effect of Hyperparameters

In terms of hyperparameters, we mainly focus on three key parameters: the number of layers, the maximum length of user/item sequences extracted from the data, and the sampling size. The number of layers helps determine the ideal size of our proposed model, while the maximum sequence length pertains to how far back in the interaction history we consider user/item interactions. Lastly, the sampling size signifies the number of steps we trace back during the message-passing process. The following Table 8 displays the selection of hyperparameters we render in our experiments:

Hyperparameters	Options
-----------------	---------

Layer numbers	1, 3, 5, 7
Max. length of sequence	5, 10, 20
Sampling size	2, 3, 4

Table 8: The tested hyperparameters in this experiment.

The experiment is conducted using the `grocery_20000` dataset, which, on average, contains approximately 27.8 interactions per graph, so it’s suitable for testing the maximum length of sequences. During the examination of each hyperparameter, the others are maintained at their default values (Table 5 in section 5.2.1). The experimental outcomes are displayed in Fig. 7.

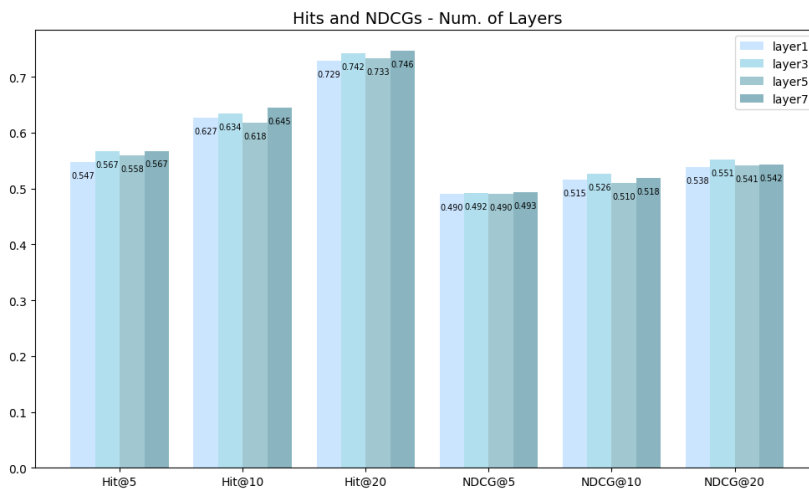


Fig. 7: The Hits and NDCGs performed on Grocery 20,000 dataset by models with different max lengths of 5, 10, 20.

The bar chart illustrates the Hit and NDCG scores for models with maximum sequence lengths of 20, 10, and 5. Interestingly, we observe that the model with a maximum length of 10 exhibits the highest performance, outperforming other max length. Conversely, the model with a maximum length of 20 consistently performs the poorest across all six evaluation metrics. Nevertheless, our findings diverge from those of Zhang [reference to section 5.4] concerning DGSR, where superior performance was observed with a maximum length of 20 compared to 10. This inconsistency can be reasonably attributed to our considerably smaller dataset. In our context, the setting of a maximum length of 20 may not effectively filter out noisy information within our graphs. Therefore, for further research on our model, we posit that the optimal max length should be determined based on alignment with the dataset we are working with, rather than adopting a fixed value like 10 or any other constant.

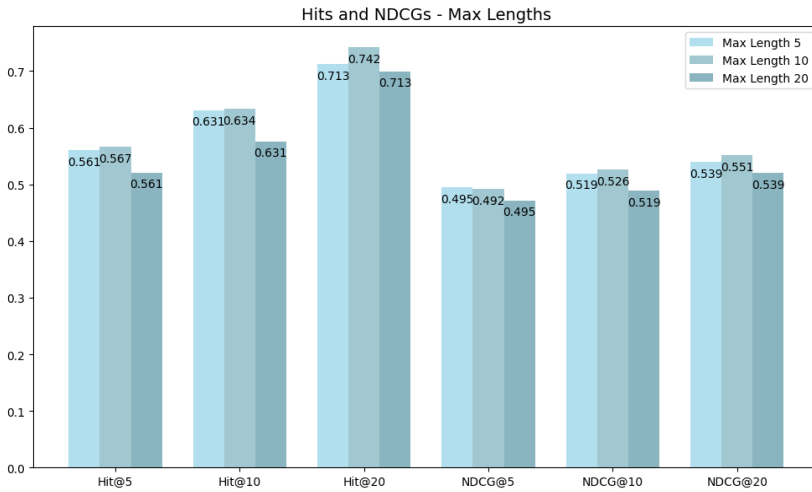


Fig. 8: The Hits and NDCGs performed on Grocery 20,000 dataset by models with 1, 3, 5, 7 DGSR layers.

The Fig. 8 visually compares the Hit and NDCG scores of models with varying DGSR layer counts: 1, 3, 5, and 7. Overall, these different layer configurations exhibit fairly similar performance. However, there is a slight upward trend in Hit scores as the number of layers increases, with a minor dip at 5 layers. Notably, models with 7 DGSR layers perform the best, followed closely by those with 3 layers. Concerning NDCG scores, the highest values consistently appear when using 3 layers, aligning with Zhang's study, which explored layer counts from 1 to 4. To summarise, employing 3 layers remains an effective choice for the Subgraph DGSR model.

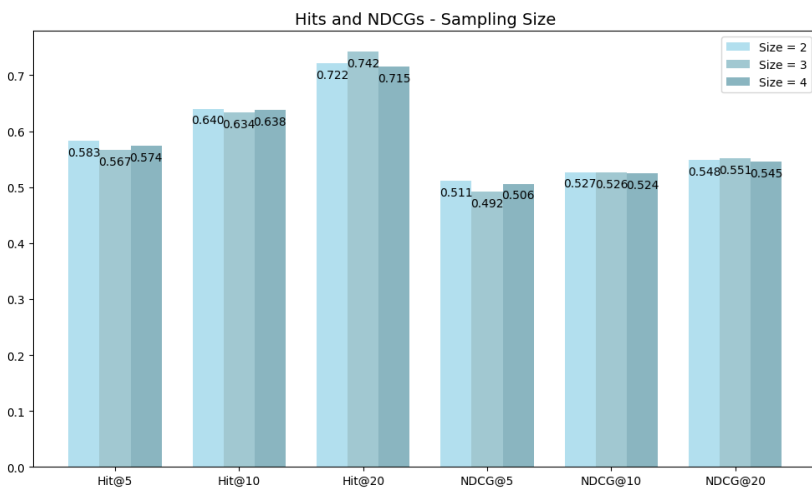


Fig. 9: The Hits and NDCGs performed on Grocery 20,000 dataset by models with sampling size of 2, 3, 4.

The overall performance of models with sizes 2, 3, and 4 shown in Fig. 9 exhibits considerable similarities, particularly in the case of Hit@10 and NDCG@10. Among these,

the size 3 model stands out with the best performance in Hit@20 and NDCG@20, despite slightly trailing the other two sizes in other scores. On the other hand, the size 2 model excels in generating accurate predictions for shorter lists (Hit@5 and NDCG@5), but struggles when tasked with predicting longer sequences. This limitation arises because a model of size 2 proves insufficient for capturing the intricate underlying patterns within the graphs required for robust performance in longer predictions. According to Zhang (2022), a larger subgraph size can offer a richer source of dynamic contextual information for each user sequence, aiding in the prediction process. Hence, if the computational complexity is affordable, we suggest using a sampling size of 3, 4, or even larger.

## RQ 3 Selection of Subgraphs

Concerning the choice of subgraphs, aside from considering their compatibility with the data, there are three additional factors that could potentially influence the composition of the subgraph lists:

- RQ 3.1 the number of subgraphs in the list;
- RQ 3.2 the subgraph size;
- RQ 3.3 the ratio of users/items.

### RQ 3.1 the number of subgraphs

In this section, we aim to assess how the number of subgraphs affects our model's performance and explanations. Additionally, we investigate the impact of introducing new structures when expanding the list of subgraphs.

Concerning the number of subgraphs, we choose to include 1, 3, and 5 subgraphs for comparison purposes. In the case of a single subgraph, we select the "Recommends" subgraph, which has displayed significant importance in our model. For the list comprising 3 subgraphs, we've utilised our default set of 3 subgraphs.

Now, for the list containing 5 subgraphs, we need to introduce 2 additional subgraphs to the default 3 subgraphs. Regarding these 2 new subgraphs, we create two versions of them. In the first version, we incorporate 2 isomorphic graphs of "Recommends" and "Multi-interests", denoted as "\*Recommends" and "\*Multi-interests", so that their structures were already included in the default 3 subgraphs, in order to minimise the impact of new structures

introduced by them. This version is referred to as "3 subgraphs + 2 duplicates." For the second version, we've designed 2 entirely new subgraphs, "2-interests" and "ShareOne," each featuring substructures not present in the initial 3 subgraphs. This version is labelled "5 subgraphs." By comparing the "3 subgraphs + 2 duplicates" and "5 subgraphs", we gain valuable insights into the choice of subgraphs to include when extending the subgraph list.

Consequently, we have four distinct lists of subgraphs, as outlined in Table 9.

<b>List1: 1 subgraph</b>		
<b>Recommends</b>		
<b>List 2: 3 subgraphs</b>		
<b>Recommends</b>	<b>Multi-interests</b>	<b>Recommends2</b>
<b>List 3: 3 subgraphs + 2 duplicates</b>		
<b>Recommends</b>	<b>Multi-interests</b>	<b>Recommends2</b>
<b>*Recommends</b>	<b>*Multi-interests</b>	

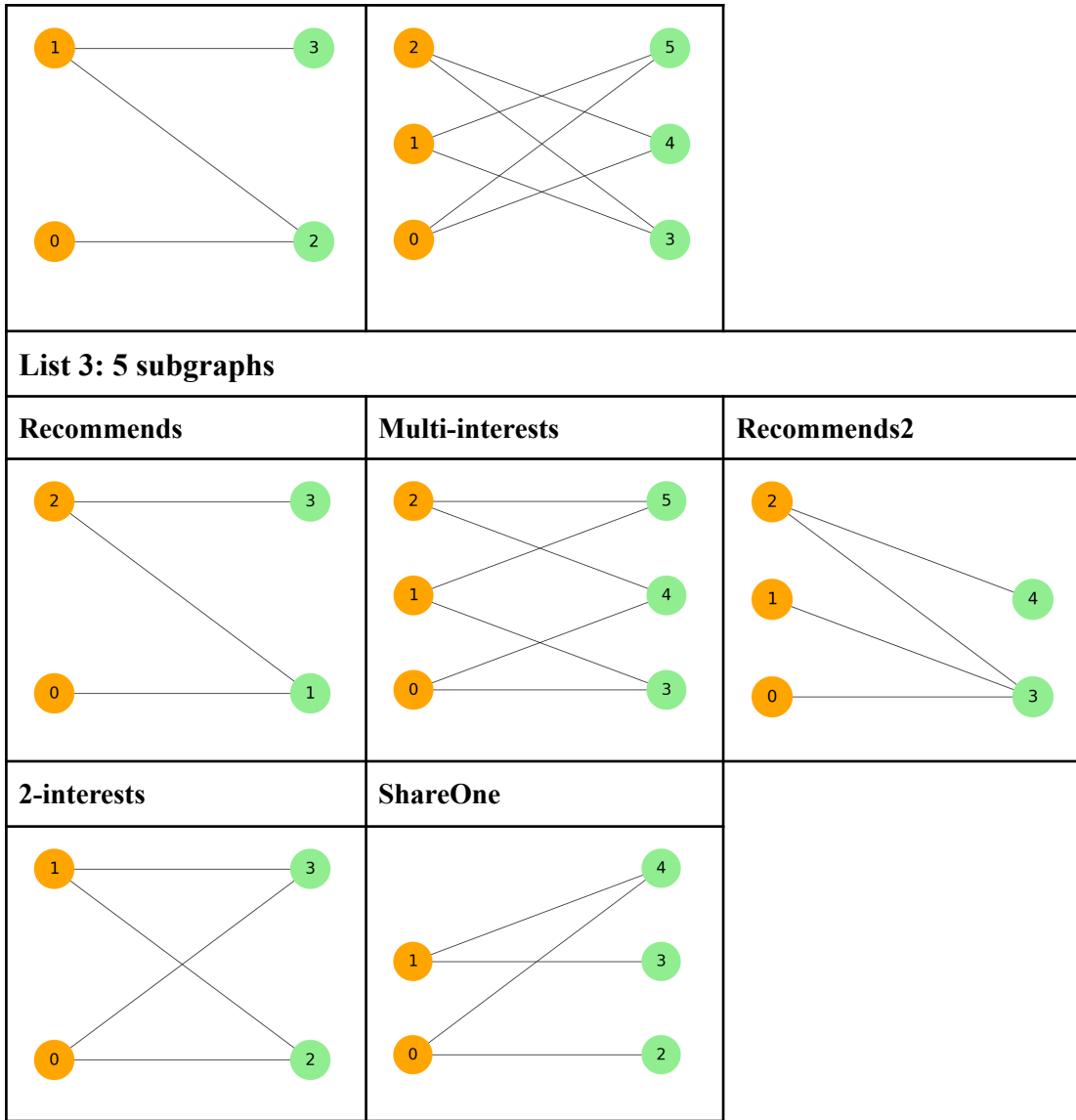


Table 9: The tested lists with 1 subgraph, 3 subgraphs, 3 subgraphs + 2 duplicates, and 5 subgraphs. User nodes are displayed in orange while item nodes are green.

The performance of models utilising lists of 1 subgraph, 3 subgraphs, 3 subgraphs +2 duplicates and 5 subgraphs is presented in Fig.10. It is evident that both Hit and NDCG scores exhibit upward trends as the number of subgraphs in the list increases (1 to 3 subgraphs, and 3 to 5 subgraphs). However, when we compare the set of 3 subgraphs to the set of 3 subgraphs plus 2 duplicates, we don't observe a significant distinction between them. This is primarily because the newly added subgraphs, "\*Recommends" and "\*Multi-interests," in the 3-subgraph list share substantial resemblance to the existing subgraphs. Consequently, they don't introduce additional structures for the model to capture, resulting in similar performance outcomes. In contrast, when adding subgraphs of new structure to the list (3 subgraphs vs. 5 subgraphs), the model performance witnesses an

obvious improvement. The test demonstrates that unique subgraphs can improve the model performance whereas duplicated subgraphs do not have the same effect.

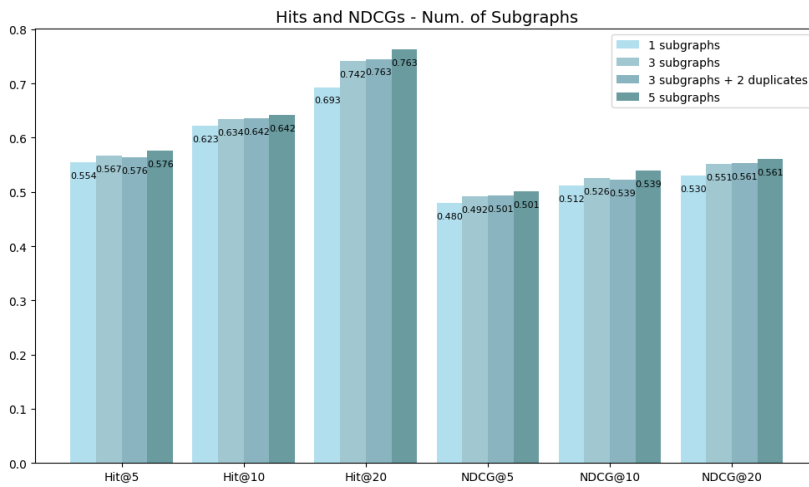


Fig. 10: The Hits and NDCGs performed on Grocery 20,000 dataset by models with 1, 3, 5 subgraphs.

Table 10 presents the importance scores of 3 subgraphs, 3 subgraphs +2 duplicates and 5 subgraphs within each model. (List1 lacks an importance score as it consists of only 1 subgraph). Among the 3 subgraphs, subgraph1 (Recommends) stands out with the highest importance score of 0.58, followed by subgraph3 (Recommends2). In the scenario of 3 subgraphs + 2 duplicates, both subgraph1 (Recommends) and subgraph4 (\*Recommends) achieve the highest importance scores, as they have identical structures. Similarly, subgraph2 (Multi-interests) and subgraph5 (\*Multi-interests) exhibit this pattern of shared importance due to their matching structures. When comparing the importance scores between "3 subgraphs" and "3 subgraphs + 2 duplicates," we observe a division of the importance scores assigned to a single subgraph in the "3 subgraphs" list into two smaller scores, allocated to the subgraph and its duplicates. This division can potentially create a misconception regarding their individual importance. However, this does not happen in the "5 subgraphs" list, where subgraph1 continues to hold the most significant share of importance, while subgraph3 and subgraph5 have distinct importance scores. This divergence is attributed to the differing structures of these subgraphs. (It's worth noting that while "Recommend" is a component of "2-interests," they are counted differently due to the symmetry of "2-interests," whereas "Recommend" lacks this symmetry.)

The test results illustrate that incorporating subgraphs with similar structures does not enhance our model's performance. Instead, it results in shared importance scores among duplicates, causing a misinterpretation of their significance. The key takeaway here is that



when selecting the list of subgraphs for our model to recognize, it's advisable to avoid including subgraphs with resembling structures and opt for unique subgraphs instead.

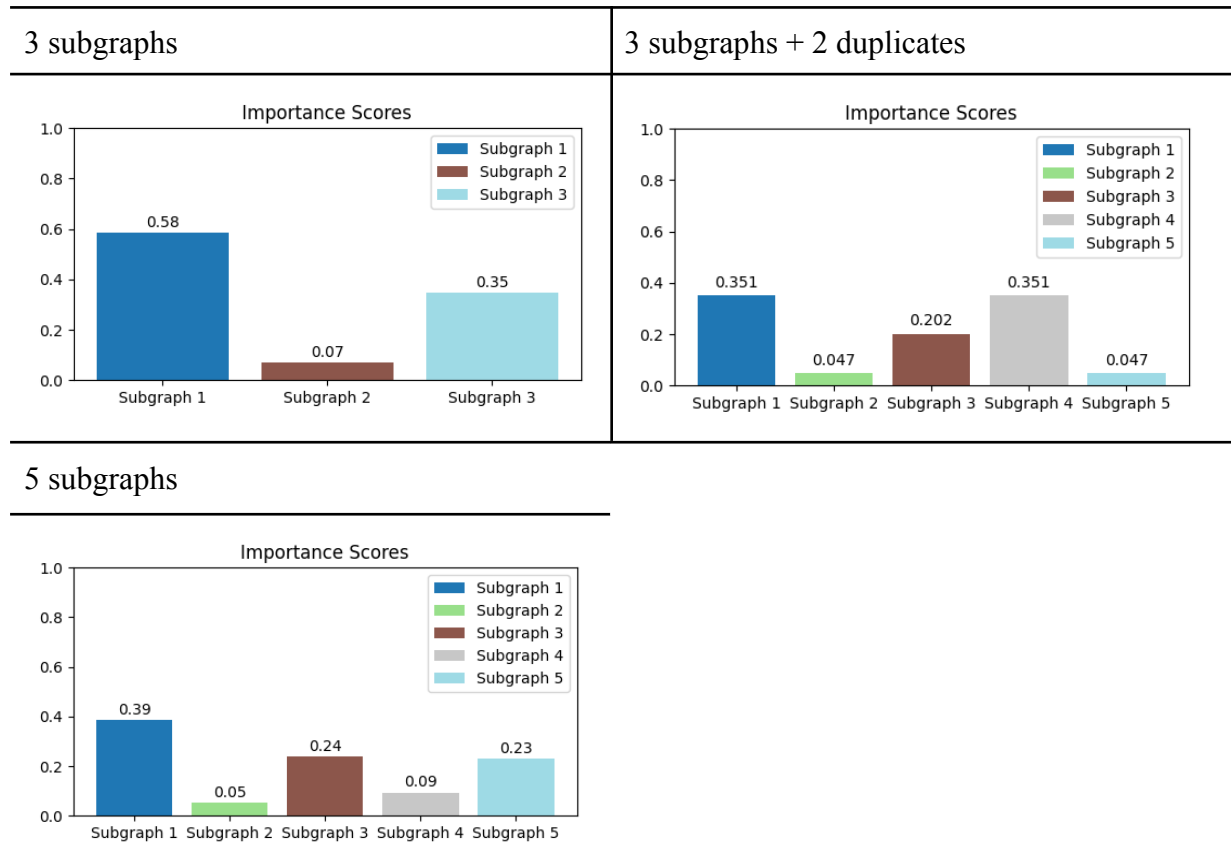


Table 10: The importance score of 3 subgraphs in list2 (left), and 5 subgraphs in list3 (right).

### RQ 3.2 the subgraph size

For the baseline subgraph list, we employ the standard set of 3 subgraphs. To assess the impact of size while maintaining other variables as constants, we choose to enlarge the dimensions of the original graph by a factor of two, as shown in Table 11. It's important to note that there are no edge connections between these enlarged dimensions. This approach ensures that when calculating the automorphisms for the subgraphs, the structure remains unaffected by any additional influences.

<b>Default 3</b>		
<b>Recommends</b>	<b>Multi-interests</b>	<b>Recommends2</b>

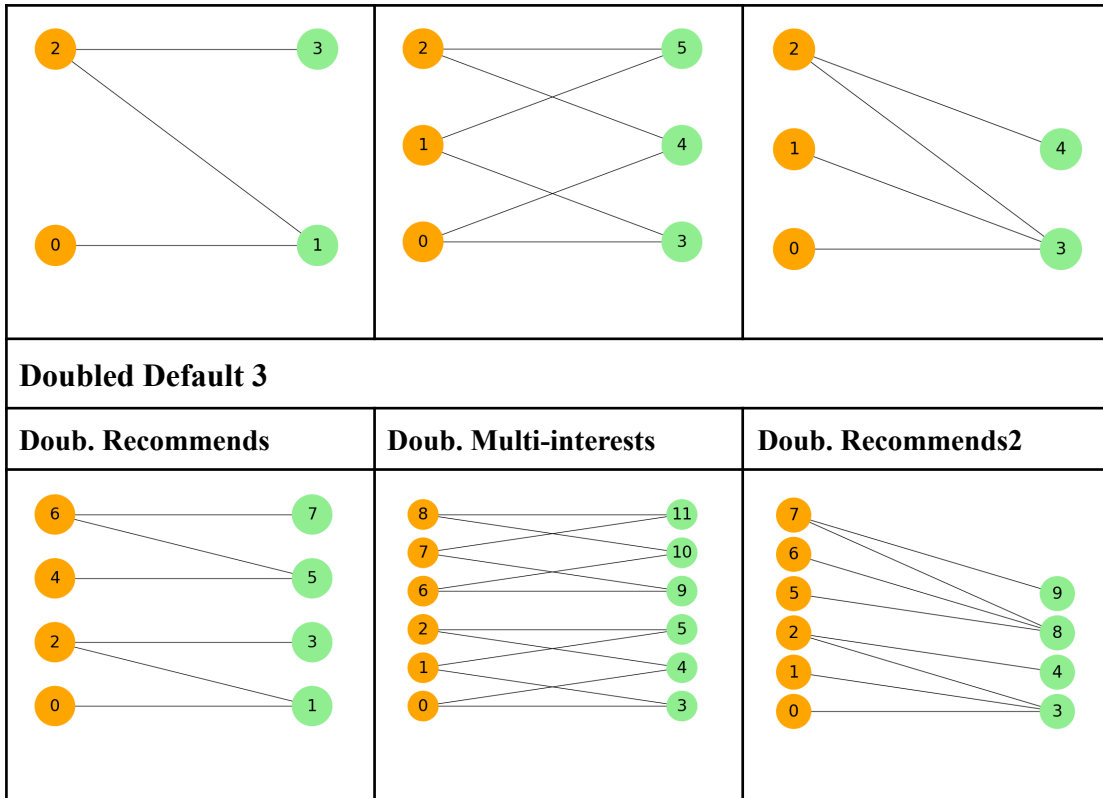


Table 11: The tested lists of default 3 and Doubled 3 subgraphs. User nodes are displayed in orange while item nodes are green.

Fig. 11 illustrates the comparative performance of models employing Default 3 subgraphs and Doubled 3 subgraphs. Minimal variation is observed across all six metrics. This consistency arises from the fact that these subgraphs share similar structures, causing our model to track identical patterns within both lists. It's worth noting that our model actually tracks two orbits with the same structure, resulting in an absence of additional information and, consequently, yielding similar outcomes. Therefore, our finding suggests that introducing larger subgraphs without significant patterns into the model does not lead to improvement on performance.

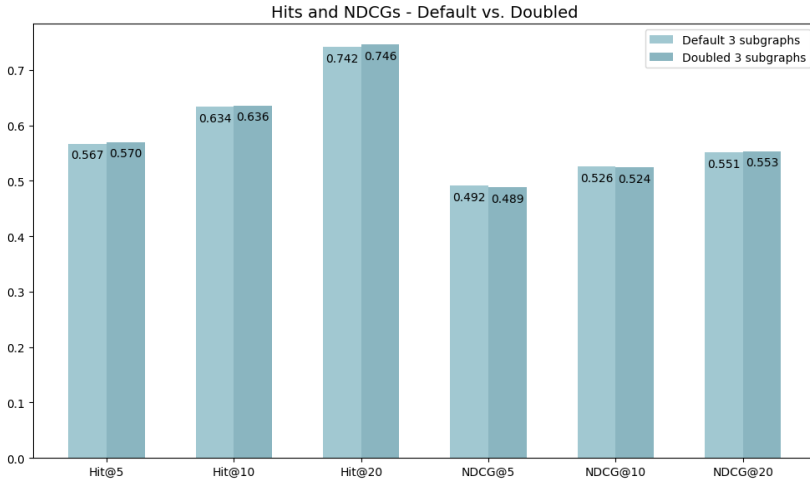


Fig. 11: The Hits and NDCGs performed on Grocery 20,000 dataset by models with Default 3 subgraphs and Doubled 3 subgraphs.

The comparison of importance scores between Default subgraphs and Doubled subgraphs in Table 12 reveals limited differences. As previously explained, the introduction of Doubled Subgraphs doesn't significantly alter the importance scores, primarily because these subgraphs exhibit similar structures. However, the disparities between subgraphs are increasing. A plausible explanation is, in the Doubled Subgraphs, two separate subgraphs are created with the same structure, the number of automorphism orbits is doubled, and so as the "popular" orbits, which makes them more easily picked up and counted by subgraph isomorphisms in the graph data.

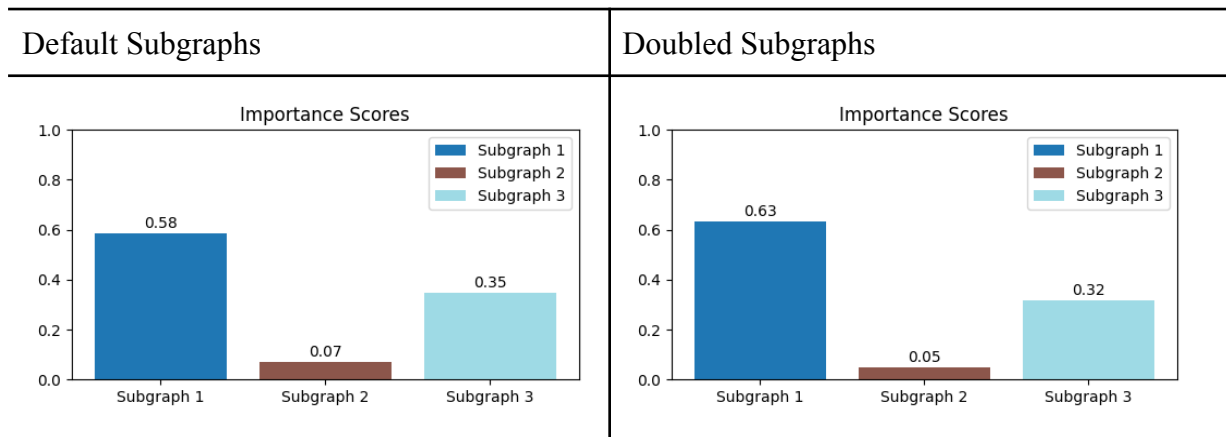


Table 12: The importance score of 3 subgraphs in list2 (left), and 5 subgraphs in list3 (right).

### RQ 3.3 the ratio of users/items

To investigate the optimal ratio of users to items to include in our model and thus to understand the design principles for subgraphs, we conducted this experiment on testing

subgraphs with different user/item ratios. We systematically adjusted the ratio by 1, 2 and 0.5, observing how these variations influenced the model's performance. The testing subgraphs are shown in Table 13. It's important to acknowledge that while the three sets of subgraphs have different user/item ratios, the structures present in List 1 are also encompassed within List 2 and List 3. In contrast, List 2 and List 3 include a few structures not found in List 1, which may inevitably contribute to higher performance for List 2 and List 3.

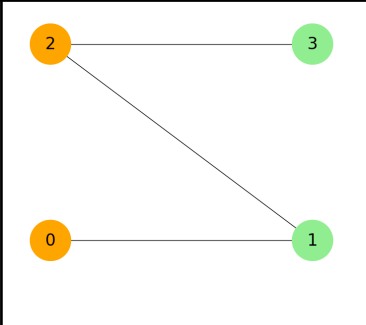
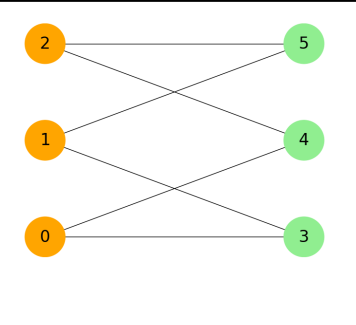
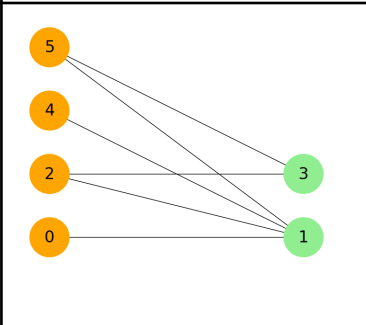
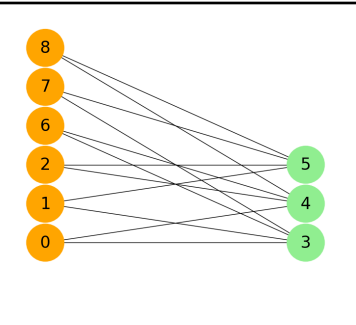
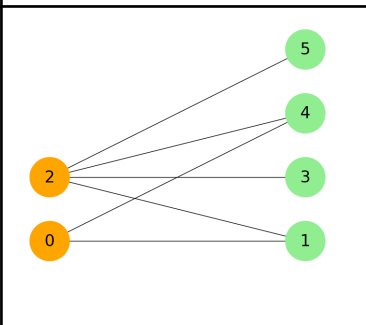
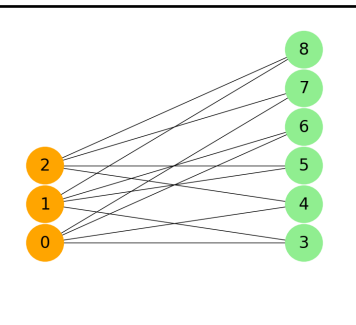
<b>List1</b>		<i>Users/Items = 1</i>	
<b>Recommends</b>		<b>Multi-interests</b>	
			
<b>List2: Double-Users</b>		<i>Users/Items = 2</i>	
<b>DU Recommends</b>		<b>DU Multi-interests</b>	
			
<b>List3: Double-Items</b>		<i>Users/Items = 0.5</i>	
<b>DI Recommends</b>		<b>DI Multi-interests</b>	
			

Table 13: The tested lists with subgraph of user/item ratio 1, 2, 0.5. User nodes are displayed in orange while item nodes are green.

Fig. 12 illustrates the performance of input subgraphs under different user/item ratios: 1, 2, and 0.5. The subgraphs where users and items are in equal proportion exhibit slightly lower compared to the other two lists with imbalanced subgraphs. That's because new structures are added to the subgraphs when doubling user or item nodes. Furthermore, there is no prominent distinction between scenarios where users outnumber items by a factor of two or where items outnumber users by a factor of two. This suggests that the user-to-item ratio typically doesn't exert a significant impact on model performance.

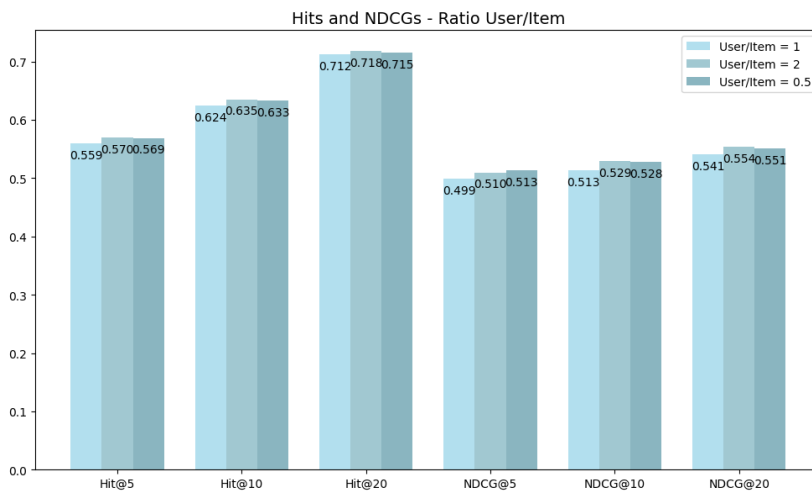
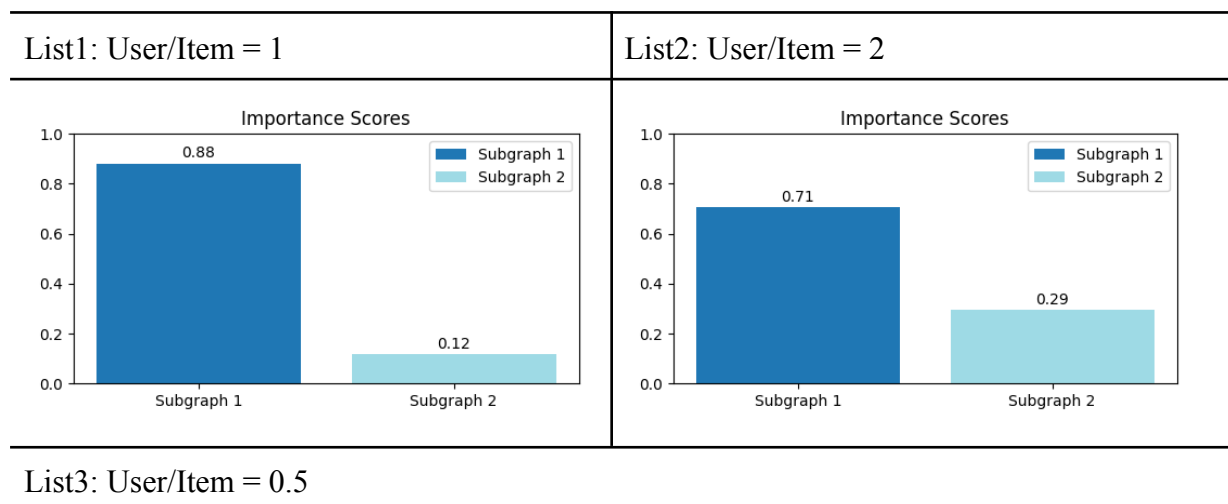


Fig. 12: The Hits and NDCGs performed on Grocery 20,000 dataset by subgraphs of user/item = 1, 2 and 0.5.

Table 14 displays the importance scores of 3 lists of subgraphs. In List 1 (User/Item = 1), there's a more pronounced gap between subgraph 1 and subgraph 2, whereas in the case of unbalanced subgraphs, the gaps between subgraph 1 and subgraph 2 tend to be narrower. This can be attributed to the fact that introducing more homogeneous orbits, and including some irrelevant ones, tends to dilute the significance of critical orbits.



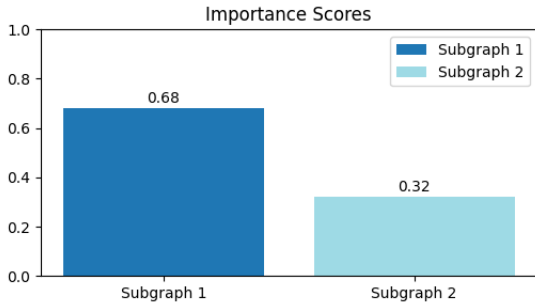


Table 14: The importance score of subgraphs of user/item = 1, 2 and 0.5.

## 6. Case Study

Campaigns in e-commerce are essential for boosting sales and user engagement. Evaluating their impact on target users can be challenging due to factors like overlapping user groups and indirect recommendation effects. To this point, our model provides a solution. By focusing on specific subgraphs and their scores, it offers clear insights into patterns of user-item interactions, including purchase, reviews and wishing list. It captures structural information within user-item networks, and thus differentiates the target groups of different campaigns. This explains why our model outperforms traditional assessment methods in depth and accuracy.

In this section, we present a case study to exemplify the framework working with our model from the perspective of an online retailer like Amazon, who is doing discount campaigns to its users. And we use another dataset, the “Home and kitchen” of Amazon within the same data source.

The data analysts of Amazon found 3 scenarios where users are more likely to respond to certain campaigns (e.g. high click through rates or redeem rates), based on surveys or exploring of their data. So they extract the structure as described in Table 15, and want our model to test if these scenarios are influential in the recommender system:

In the first scenario “Recommend 1”, the analysts find that, when user1 buys a cooker (item5), a coupon would trigger him to buy bowls, plates and tissue (item2, 3, 4) as user 0 did. They highlight this scenario and want to know if this scene generally takes place, and whether this can happen on other combinations of products by other users. Consequently, this scenario is recorded as the first subgraph. Similarly, in “Recommend 2”, analysts find that user1 responds to coupons while user2 responds to cross-selling discounts, regarding item3,

4, 5, and to the same point, the analysts take down the scenarios. For comparison, they input the scenario where users hardly respond to promotions, represented as “Flexible”. With the collaboration of the campaign team, they design corresponding strategies to the 3 scenarios.

(Please note, the users and items are not specific ones. In our model, our subgraph mechanism looks onto the entire data for the same scenarios (structures) as subgraphs, that exist in our data, regardless of certain users and items, while DGSR proceeds these structures onto various combinations of specific users and items.)

Scenarios		
Recommend 1	Recommend 2	Flexible
Targeting Campaigns - Example		
1. Coupons to user1 on items 3, 4, 5.	2. Coupons to user1 on item 3, 4, 5; discounts to user2 when buying item7 with item3, 4, 5.	3. No promotions

Table 15: The subgraphs of different scenarios aligned with campaigns targeting them. User nodes are displayed in orange while item nodes are green.

After conducting exploratory data analysis, the data analysts observed that the average purchase length in the Home and Kitchen category was approximately 9.1, similar to Grocery and shorter than the CD and Music datasets. As a result, they opted to set the max\_length hyperparameter to 10 for the model, while retaining the default settings for other hyperparameters. Following training and testing on the dataset, the data analysts obtained a model with the performance illustrated in Fig. 13, and importance scores shown in Fig. 14.

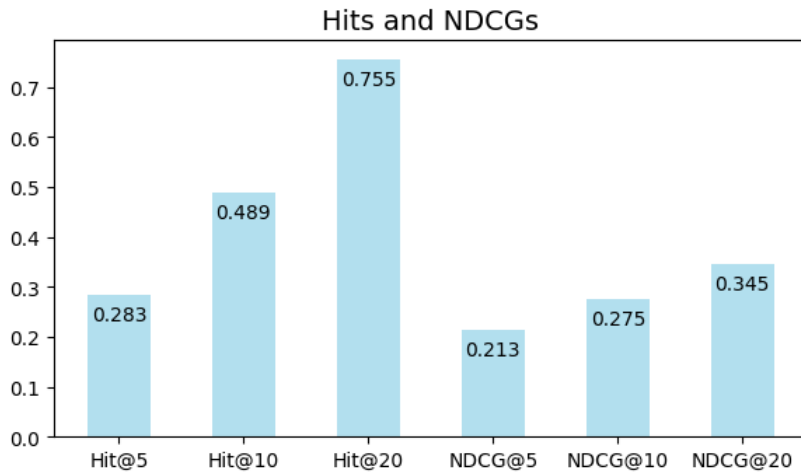


Fig. 13: The performance of the model training on Home dataset.

The obvious increasing trends in recommendation performance as the predicted list length increases are attributed to the fact that items that are relatively more relevant appear later in the predicted list. This suggests that our model, when applied to the Home and Kitchen dataset, excels in generating recommendations with a large number of options available for users to choose from.

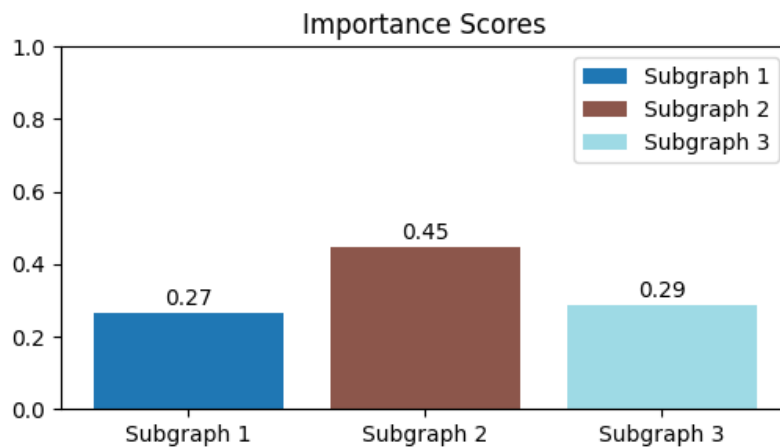


Fig. 14: The importance score of subgraphs of user/item = 1, 2 and 0.5.

When generating recommendations, the impact of subgraph2 (representing "Recommend 2") is the most significant, contributing to predictions with a weight of 0.45. Subsequently, subgraph3, representing "Flexible", plays a relatively less influential role, contributing around 0.29, closely followed by subgraph1 "Recommend 1".

Thus, in terms of campaign strategy selection, the analysts prioritise the strategy: when a "Recommend 2" scenario is detected, they provide coupons to user1 on item 3, 4, 5 and discounts to user2 when buying item7 together with one of item 3, 4, 5. As the "Recommend



1" scenario holds the least importance, the campaign team takes no actions on that group of potential buyers. In general, based on the model results, the analysts decide to carry on targeting campaign 2. They also utilise our model to predict recommendations for users and display promotions to them if the "Recommend 2" scenario is detected on them.

The case study can be expanded to encompass more detailed campaign designs, incorporate additional user group subgraphs as input, and consider a broader range of user/item interactions such as views, subscriptions, and ratings. This approach aims to finely tailor campaign strategies to specific user groups, ensuring more precise targeting and potentially higher effectiveness.

## 7. Discussion

In current research, we have successfully developed a structure-enhanced and explainable Graph Neural Network (GNN) model for recommendation systems. Our model demonstrates commendable strengths in recommendation accuracy and interpretability, offering importance scores of personalised subgraphs which patterns the model extracts from user-item interactions and thus enhancing the overall quality of recommendations.

Throughout our study, we highlight the following main findings:

1. Our Subgraph DGSR model demonstrates superior performance compared to the original DGSR when configured with appropriate settings and meaningful subgraphs (RQ1);
2. For future research with our model, we recommend a thorough exploration of the dataset and computational resources to determine optimal hyperparameters;
3. The selection of subgraph lists plays a crucial role in our Subgraph DGSR model, especially the inclusion of explanatory structures that are prevalent in the training and testing data.
  - a. It's essential to diversify the subgraph structures when expanding the list of subgraphs for the model to focus on;
  - b. Introducing larger subgraphs without significant patterns into the model does not directly result in improved performance;

- c. There is no compelling evidence to suggest that managing the user/item ratio is necessary to achieve better model performance and higher importance scores.

However, despite its achievements, certain limitations need to be addressed to fully realise its potential and applicability in real-world scenarios:

One of the primary drawbacks of our model is its inherent inflexibility in modifying tracking subgraphs once the training process begins. This limitation poses challenges for adaptability, especially when dynamic changes occur in user preferences and product behaviours over time. To overcome this drawback and bolster the model's responsiveness to evolving data, future research endeavours can explore the integration of transfer learning techniques. By incorporating transfer learning, we can facilitate fine-tuning the model on new or updated tracking subgraphs, enabling it to accommodate changes without necessitating complete retraining. This improvement would undoubtedly enhance the model's practicality and make it more suited for real-time recommendation scenarios.

Moreover, the generalizability of our model warrants continued investigation. While our experiments showcased promising results on the Amazon user review dataset, conducting evaluations on a broader spectrum of datasets from various domains would provide a comprehensive understanding of the model's ability to adapt across diverse recommendation scenarios. By subjecting the model to testing on multiple datasets, we can assess its sensitivity to dataset-specific characteristics and gain insights into potential areas for refinement. This thorough assessment would instil greater confidence in the model's applicability and reliability, cementing its position as a versatile recommendation solution.

In conclusion, our study has contributed to the field of explainable recommendation systems by presenting a robust and interpretable GNN model. We have highlighted both its strengths and limitations, acknowledging the need for further developments to realise its full potential. The application of transfer learning to address tracking subgraph constraints and the comprehensive evaluation of the model on diverse datasets represent promising avenues for future research. As we move forward, refining and expanding the capabilities of our model will undoubtedly elevate its practical impact, empowering online retailers and businesses to deliver accurate, transparent, and personalised recommendations, thereby enriching the user experience and fostering stronger customer relationships.

# Reference

Bouritsas, G., Frasca, F., Zafeiriou, S., & Bronstein, M. M. (2022). Improving graph neural network expressivity via subgraph isomorphism counting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(1), 657-668.

Chang, J., Gao, C., He, X., Jin, D., & Li, Y. (2020, July). Bundle recommendation with graph convolutional networks. In *Proceedings of the 43rd international ACM SIGIR conference on Research and development in Information Retrieval* (pp. 1673-1676).

Chen, C., Guo, J., & Song, B. (2021, July). Dual attention transfer in session-based recommendation with multi-dimensional integration. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval* (pp. 869-878).

Chen, T., & Wong, R. C. W. (2020, August). Handling information loss of graph neural networks for session-based recommendation. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (pp. 1172-1180).

Chen, J., Wang, X., & Xu, X. (2022). GC-LSTM: Graph convolution embedded LSTM for dynamic network link prediction. *Applied Intelligence*, 1-16.

Chang, J., Gao, C., Zheng, Y., Hui, Y., Niu, Y., Song, Y., ... & Li, Y. (2021, July). Sequential recommendation with graph neural networks. In *Proceedings of the 44th international ACM SIGIR conference on research and development in information retrieval* (pp. 378-387).

Cordella, L. P., Foggia, P., Sansone, C., & Vento, M. (2001, May). An improved algorithm for matching large graphs. In *3rd IAPR-TC15 workshop on graph-based representations in pattern recognition* (pp. 149-159).

Covington, P., Adams, J., & Sargin, E. (2016, September). Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM conference on recommender systems* (pp. 191-198).

Feng, Y., You, H., Zhang, Z., Ji, R., & Gao, Y. (2019, July). Hypergraph neural networks. In *Proceedings of the AAAI conference on artificial intelligence* (Vol. 33, No. 01, pp. 3558-3565).

Gao, C., Wang, X., He, X., & Li, Y. (2022, February). Graph neural networks for recommender system. In *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining* (pp. 1623-1625).

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., & Dahl, G. E. (2017, July). Neural message passing for quantum chemistry. In *International conference on machine learning* (pp. 1263-1272). PMLR.

Goyal, P., Kamra, N., He, X., & Liu, Y. (2018). Dyngem: Deep embedding method for dynamic graphs. *arXiv preprint arXiv:1805.11273*.

Guo, L., Tang, L., Chen, T., Zhu, L., Nguyen, Q. V. H., & Yin, H. (2021). DA-GCN: A domain-aware attentive graph convolution network for shared-account cross-domain sequential recommendation. *arXiv preprint arXiv:2105.03300*.

- Gupta, P., Garg, D., Malhotra, P., Vig, L., & Shroff, G. (2019). NISER: Normalized item and session representations to handle popularity bias. arXiv preprint arXiv:1909.04276.
- Hamilton, W., Ying, Z., & Leskovec, J. (2017). Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30.
- He, X., Deng, K., Wang, X., Li, Y., Zhang, Y., & Wang, M. (2020, July). 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* (pp. 639-648).
- Huang, H., Fang, Z., Wang, X., Miao, Y., & Jin, H. (2020, July). Motif-Preserving Temporal Network Embedding. In *IJCAI* (pp. 1237-1243).
- Jin, B., Gao, C., He, X., Jin, D., & Li, Y. (2020, July). Multi-behavior recommendation with graph convolutional networks. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval* (pp. 659-668).
- Kang, W. C., & McAuley, J. (2018, November). Self-attentive sequential recommendation. In *2018 IEEE international conference on data mining (ICDM)* (pp. 197-206). IEEE.
- Kipf, T. N., & Welling, M. (2016). Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907.
- Koren, Y. (2008, August). Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 426-434).
- Kumar, S., Zhang, X., & Leskovec, J. (2019, July). Predicting dynamic embedding trajectory in temporal interaction networks. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining* (pp. 1269-1278).
- Li, J., Dani, H., Hu, X., Tang, J., Chang, Y., & Liu, H. (2017, November). Attributed network embedding for learning in a dynamic environment. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management* (pp. 387-396).
- Liu, J., Kawaguchi, K., Hooi, B., Wang, Y., & Xiao, X. (2021). Eignn: Efficient infinite-depth graph neural networks. *Advances in Neural Information Processing Systems*, 34, 18762-18773.
- Li, G., Liu, H., Li, G., Shen, S., & Tang, H. (2020). LSTM-based argument recommendation for non-API methods. *Science China Information Sciences*, 63, 1-22.
- Li, Y., Lu, L., & Xuefeng, L. (2005). A hybrid collaborative filtering method for multiple-interests and multiple-content recommendation in E-Commerce. *Expert systems with applications*, 28(1), 67-77.
- Liu, X., & Song, Y. (2022, June). Graph convolutional networks with dual message passing for subgraph isomorphism counting and matching. In *Proceedings of the AAAI Conference on Artificial Intelligence* (Vol. 36, No. 7, pp. 7594-7602).
- Lin, C., Sun, G. J., Bulusu, K. C., Dry, J. R., & Hernandez, M. (2020). Graph neural networks including sparse interpretability. arXiv preprint arXiv:2007.00119.

- Li, Z., Shen, X., Jiao, Y., Pan, X., Zou, P., Meng, X., ... & Bu, J. (2020, April). Hierarchical bipartite graph neural networks: Towards large-scale e-commerce applications. In 2020 IEEE 36th International Conference on Data Engineering (ICDE) (pp. 1677-1688). IEEE.
- Li, Y., Tarlow, D., Brockschmidt, M., & Zemel, R. (2015). Gated graph sequence neural networks. arXiv preprint arXiv:1511.05493.
- Li, J., Wang, Y., & McAuley, J. (2020, January). Time interval aware self-attention for sequential recommendation. In Proceedings of the 13th international conference on web search and data mining (pp. 322-330).
- Li, Y., Zhou, J., Verma, S., & Chen, F. (2022). A survey of explainable graph neural networks: Taxonomy and evaluation metrics. arXiv preprint arXiv:2207.12599.
- Maron, H., Ben-Hamu, H., Serviansky, H., & Lipman, Y. (2019). Provably powerful graph networks. Advances in neural information processing systems, 32.
- Maron, H., Ben-Hamu, H., Shamir, N., & Lipman, Y. (2018). Invariant and equivariant graph networks. arXiv preprint arXiv:1812.09902.
- Maron, H., Fetaya, E., Segol, N., & Lipman, Y. (2019, May). On the universality of invariant networks. In International conference on machine learning (pp. 4363-4371). PMLR.
- Morris, C., Ritzert, M., Fey, M., Hamilton, W. L., Lenssen, J. E., Rattan, G., & Grohe, M. (2019, July). Weisfeiler and leman go neural: Higher-order graph neural networks. In Proceedings of the AAAI conference on artificial intelligence (Vol. 33, No. 01, pp. 4602-4609).
- Ni, J., Li, J., & McAuley, J. (2019). Empirical Methods in Natural Language Processing (EMNLP).
- Qiu, R., Li, J., Huang, Z., & Yin, H. (2019, November). Rethinking the item order in session-based recommendation with graph neural networks. In Proceedings of the 28th ACM international conference on information and knowledge management (pp. 579-588).
- Qu, L., Zhu, H., Duan, Q., & Shi, Y. (2020, April). Continuous-time link prediction via temporal dependent graph neural network. In Proceedings of The Web Conference 2020 (pp. 3026-3032).
- Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P., & Riedl, J. (1994, October). Grouplens: An open architecture for collaborative filtering of netnews. In Proceedings of the 1994 ACM conference on Computer supported cooperative work (pp. 175-186).
- Strub, F., Mary, J., & Gaudel, R. (2016). Hybrid collaborative filtering with autoencoders. arXiv preprint arXiv:1603.00806.
- Sankar, A., Wu, Y., Gou, L., Zhang, W., & Yang, H. (2020, January). Dysat: Deep neural representation learning on dynamic graphs via self-attention networks. In Proceedings of the 13th international conference on web search and data mining (pp. 519-527).
- Skarding, J., Gabrys, B., & Musial, K. (2021). Foundations and modeling of dynamic networks using dynamic graph neural networks: A survey. IEEE Access, 9, 79143-79168.
- Tiago P. Peixoto, "The graph-tool python library", figshare. (2014) DOI: 10.6084/m9.figshare.1164194 [sci-hub, @tor]

- Trivedi, R., Farajtabar, M., Biswal, P., & Zha, H. (2019, May). Dyrep: Learning representations over dynamic graphs. In International conference on learning representations.
- Velickovic, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., & Bengio, Y. (2017). Graph attention networks. *stat*, 1050(20), 10-48550.
- Vu, M., & Thai, M. T. (2020). Pgm-explainer: Probabilistic graphical model explanations for graph neural networks. *Advances in neural information processing systems*, 33, 12225-12235.
- Weisfeiler, B., & Leman, A. (1968). The reduction of a graph to canonical form and the algebra which appears therein. *nti, Series*, 2(9), 12-16.
- Wu, L., Sun, P., Fu, Y., Hong, R., Wang, X., & Wang, M. (2019, July). A neural influence diffusion model for social recommendation. In Proceedings of the 42nd international ACM SIGIR conference on research and development in information retrieval (pp. 235-244).
- Wu, S., Sun, F., Zhang, W., Xie, X., & Cui, B. (2022). Graph neural networks in recommender systems: a survey. *ACM Computing Surveys*, 55(5), 1-37.
- Wang, Z., Wei, W., Cong, G., Li, X. L., Mao, X. L., & Qiu, M. (2020, July). Global context enhanced graph neural networks for session-based recommendation. In Proceedings of the 43rd international ACM SIGIR conference on research and development in information retrieval (pp. 169-178).
- Wang, W., Zhang, W., Liu, S., Liu, Q., Zhang, B., Lin, L., & Zha, H. (2020, April). Beyond clicks: Modeling multi-relational item graph for session-based target behavior prediction. In Proceedings of the web conference 2020 (pp. 3056-3062).
- Xu, K., Hu, W., Leskovec, J., & Jegelka, S. (2018). How powerful are graph neural networks?. *arXiv preprint arXiv:1810.00826*.
- Xie, X., Liu, Z., Wu, S., Sun, F., Liu, C., Chen, J., ... & Ding, B. (2021, October). Causcf: Causal collaborative filtering for recommendation effect estimation. In Proceedings of the 30th ACM International Conference on Information & Knowledge Management (pp. 4253-4263).
- Xu, D., Ruan, C., Korpeoglu, E., Kumar, S., & Achan, K. (2020). Inductive representation learning on temporal graphs. *arXiv preprint arXiv:2002.07962*.
- Ying, Z., Bourgeois, D., You, J., Zitnik, M., & Leskovec, J. (2019). Gnnexplainer: Generating explanations for graph neural networks. *Advances in neural information processing systems*, 32.
- Ying, R., He, R., Chen, K., Eksombatchai, P., Hamilton, W. L., & Leskovec, J. (2018, July). Graph convolutional neural networks for web-scale recommender systems. In Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining (pp. 974-983).
- Yuan, H., Yu, H., Gui, S., & Ji, S. (2022). Explainability in graph neural networks: A taxonomic survey. *IEEE transactions on pattern analysis and machine intelligence*, 45(5), 5782-5799.
- Zhang, Y., Defazio, D., Ramesh, A.: Relex: A model-agnostic relational model explainer. *arXiv preprint arXiv:2006.00305* (2020)

Zhang, M., Wu, S., Gao, M., Jiang, X., Xu, K., & Wang, L. (2020). Personalized graph neural networks with attention mechanism for session-aware recommendation. *IEEE Transactions on Knowledge and Data Engineering*, 34(8), 3946-3957.

Zhang, M., Wu, S., Yu, X., Liu, Q., & Wang, L. (2022). Dynamic graph neural networks for sequential recommendation. *IEEE Transactions on Knowledge and Data Engineering*, 35(5), 4741-4753.

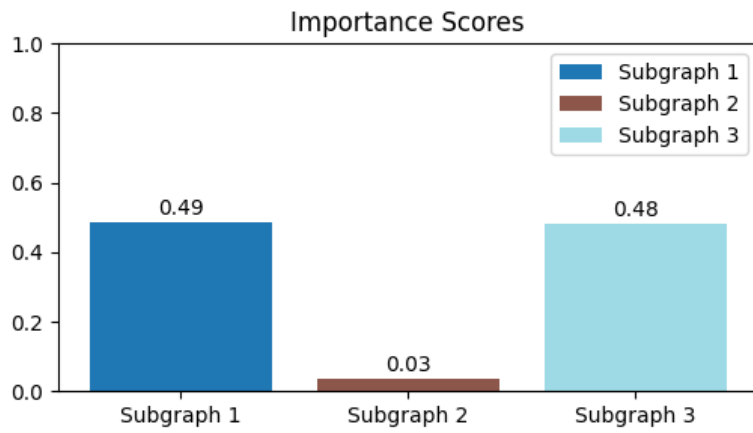
Zhou, L., Yang, Y., Ren, X., Wu, F., & Zhuang, Y. (2018, April). Dynamic network embedding by modeling triadic closure process. In *Proceedings of the AAAI conference on artificial intelligence* (Vol. 32, No. 1).

Zhang, S., Yao, L., Sun, A., & Tay, Y. (2019). Deep learning based recommender system: A survey and new perspectives. *ACM computing surveys (CSUR)*, 52(1), 1-38.

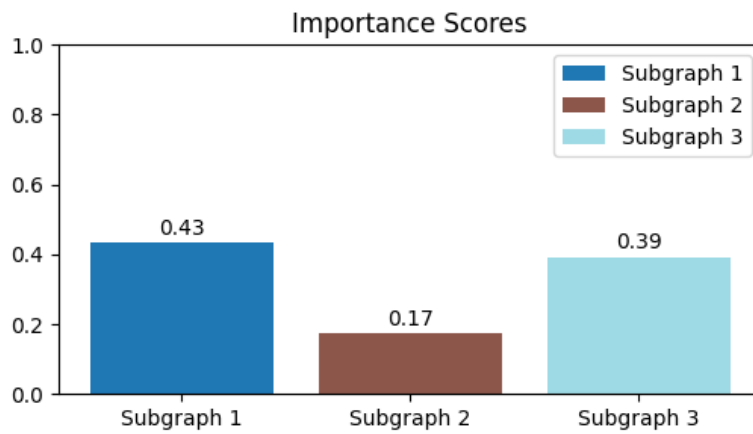
Zuo, Y., Liu, G., Lin, H., Guo, J., Hu, X., & Wu, J. (2018, July). Embedding temporal network via neighborhood formation. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining* (pp. 2857-2866).

# Appendix 1: Importance Scores of RQ1

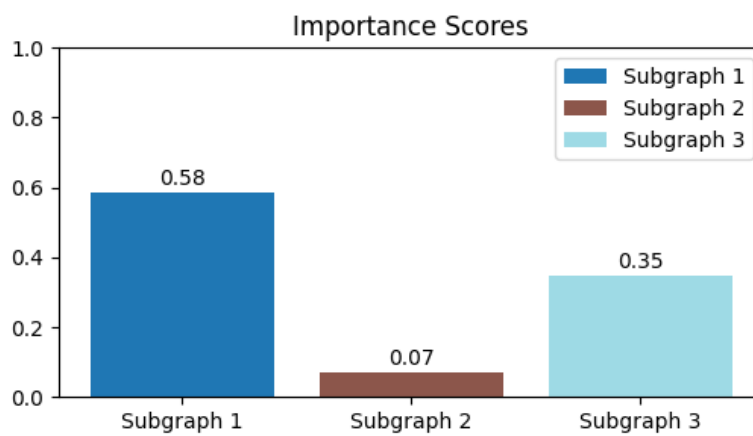
CD



Music



Grocery





## Appendix 2: Script of Subgraph Isomorphism

```
import dgl
import torch
import numpy as np
from torch_geometric.utils import remove_self_loops, to_undirected
import networkx as nx
from networkx.algorithms import isomorphism

import networkx as nx
import matplotlib.pyplot as plt
import numpy as np

def automorphism_orbits(edge_list, print_msgs=True, **kwargs):

    ##### vertex automorphism orbits #####

    directed = kwargs['directed'] if 'directed' in kwargs else False

    G_nx = nx.from_edgelist(edge_list, create_using=nx.DiGraph() if directed else nx.Graph())
    G_nx.remove_edges_from(nx.selfloop_edges(G_nx))

    matcher = isomorphism.GraphMatcher(G_nx, G_nx)
    aut_group = [iso for iso in matcher.subgraph_isomorphisms_iter()]

    orbit_membership = {v: v for v in G_nx.nodes()}

    for aut in aut_group:
        for original, vertex in aut.items():
            role = min(original, orbit_membership[vertex])
            orbit_membership[vertex] = role

    # make orbit list contiguous (i.e. 0,1,2,...O)
    unique_orbits = list(set(orbit_membership.values()))
    contiguous_orbit_membership = {orbit: idx for idx, orbit in enumerate(unique_orbits)}

    orbit_partition = {}
    for vertex, orbit in orbit_membership.items():
        if contiguous_orbit_membership[orbit] not in orbit_partition:
            orbit_partition[contiguous_orbit_membership[orbit]] = []
        orbit_partition[contiguous_orbit_membership[orbit]].append(vertex)

    aut_count = len(aut_group)

    if print_msgs:
        print('Orbit partition of given substructure: {}'.format(orbit_partition))
        print('Number of orbits: {}'.format(len(orbit_partition)))
        print('Automorphism count: {}'.format(aut_count))

    return G_nx, orbit_partition, orbit_membership, aut_count

def subgraph_isomorphism_edge_counts(edge_index, num_edge, **kwargs):
```

```

##### edge structural identifiers #####

subgraph_dict, induced = kwargs['subgraph_dict'], kwargs['induced']
directed = kwargs['directed'] if 'directed' in kwargs else False

# Convert edge_index to NetworkX graph
G_nx = nx.from_edgelist(edge_index.transpose(1,0).cpu().numpy(), create_using=nx.DiGraph() if
directed else nx.Graph())
subgraph = subgraph_dict['subgraph']

# Check for isomorphism
GM = isomorphism.GraphMatcher(G_nx, subgraph)
sub_isos = [iso for iso in GM.subgraph_isomorphisms_iter()]

edge_dict = {tuple(edge): i for i, edge in enumerate(edge_index.transpose(1,0).cpu().numpy())}
subgraph_edges = list(subgraph.edges())

orbit_len = sum(len(lst) for lst in subgraph_dict['orbit_partition'].values())
counts = np.zeros((num_edge, orbit_len))

#print("-" * 20)
#print("counts size:", counts.shape)
#print(subgraph_dict['orbit_partition'])

for sub_iso in sub_isos:
    for i, edge in enumerate(subgraph_edges):
        # Ensure both nodes of the edge are in the mapping
        if edge[0] in sub_iso and edge[1] in sub_iso:

            edge_orbit = subgraph_dict['orbit_membership'][i]

            mapped_edge = (sub_iso[edge[0]], sub_iso[edge[1]])
            if mapped_edge in edge_dict:

                counts[edge_dict[mapped_edge], edge_orbit] += 1
                # counts[edge_dict[mapped_edge], i] += 1

counts = counts / subgraph_dict['aut_count']
return torch.tensor(counts)

def subgraph_counts2ids(data, subgraph_dicts, subgraph_params):

    # Assuming a single edge type for simplicity
    edge_type = data.canonical_etypes[0]
    # Retrieve edges

    src, dst = data.edges(form='uv', etype=edge_type)
    edge_index = torch.stack([src, dst], dim=0)

    num_edge_by = edge_index.shape[1]

    if 'edge_features' in data.edata[edge_type]:
        edge_features_dict = data.edata[edge_type]['edge_features']

```

```

# If edge_features is a dictionary, process each tensor in the dictionary
if isinstance(edge_features_dict, dict):
    for key, value in edge_features_dict.items():
        if not isinstance(value, torch.Tensor):
            value = torch.tensor(value)
            edge_index, value = remove_self_loops(edge_index, value)
            edge_features_dict[key] = value
        data.edata[edge_type]['edge_features'] = edge_features_dict
    else:
        edge_index, edge_features = remove_self_loops(edge_index, edge_features_dict)
        data.edata[edge_type]['edge_features'] = edge_features
else:
    edge_index = remove_self_loops(edge_index)[0]

num_nodes = data.number_of_nodes()
identifiers = None
for subgraph_dict in subgraph_dicts:
    kwargs = {
        'subgraph_dict': subgraph_dict,
        'induced': subgraph_params['induced'],
        'num_nodes': num_nodes,
        'directed': subgraph_params['directed']
    }

    counts = subgraph_isomorphism_edge_counts(edge_index, num_edge = num_edge_by,
**kwargs)

    identifiers = counts if identifiers is None else torch.cat((identifiers, counts), 1)

# Store the computed identifiers as edge data
data.edges['by'].data['identifiers'] = identifiers.long()

#print(identifiers.long().shape)

return data

```