Explaining GNN-based Recommendations in Logic

Wenfei Fan
Beihang University, China
Shenzhen Institute of Computing Sciences, China
University of Edinburgh
United Kingdom
wenfei@inf.ed.ac.uk

Lihang Fan
Beihang University, China
fanlh@buaa.edu.cn

Dandan Lin*
Shenzhen Institute of Computing Sciences, China
lindandan@sics.ac.cn

Min Xie
Shenzhen Institute of Computing Sciences, China
xiemin@sics.ac.cn

ABSTRACT

This paper proposes Makex (MAKE senSE), a logic approach to explaining why a GNN-based model $\mathcal{M}(x, y)$ recommends item y to user x. It proposes a class of Rules for ExPlanations, denoted as REPs and defined with a graph pattern Q and dependency $X \rightarrow \mathcal{M}(x, y)$, where X is a collection of predicates, and the model $\mathcal{M}(x, y)$ is treated as the consequence of the rule. Intuitively, given $\mathcal{M}(x, y)$, we discover pattern Q to identify relevant topology, and precondition X to disclose correlations, interactions and dependencies of vertex features; together they provide rationals behind prediction $\mathcal{M}(x, y)$, identifying what features are decisive for $\mathcal{M}$ to make predictions and under what conditions the decision can be made. We (a) define REPs with 1-WL test, on which most GNN models for recommendation are based; (b) develop an algorithm for discovering REPs for $\mathcal{M}$ as global explanations, and (c) provide a top-k algorithm to compute top-ranked local explanations. Using real-life graphs, we empirically verify that Makex outperforms previous explanation methods in terms of fidelity, sparsity and efficiency.

PVLDB Reference Format:
Wenfei Fan, Lihang Fan, Dandan Lin, Min Xie. Explaining GNN-based Recommendations in Logic. PVLDB, 18(3): 715 - 728, 2024.
doi:10.14778/3712221.3712237

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/SICS-Fundamental-Research-Center/Makex.

1 INTRODUCTION

Graph neural networks (GNNs) have found prevalent use in recommender systems since they accurately model user preferences from historical user-item interactions by exploring multi-hop relationships between users and items in graph-structured data [29, 45, 81]. A variety of GNN-based recommendation models have been trained *e.g.,* [13, 15, 16, 19, 34, 37, 39, 40, 44, 46, 47, 52, 62, 67, 71, 73, 75, 78, 81] (surveyed in [29, 76]), and deployed at *e.g.,* Pinterest [81], Tencent [45, 87], Alibaba [65], Amazon [3, 4, 48] and Uber [6].

With this comes the need for explaining GNN-based recommendations $\mathcal{M}(x, y)$, to tell why an item y is recommended to user x. The reason is twofold, (a) to provide the users with insights and establish their trust in the predictions [42, 43, 57], and (b) help developers debug ML models by revealing errors or bias in training data that result in adverse and unexpected behaviors [50].

Explaining GNN-based predictions has been approached as follows: (a) self-explainable GNN models build explanations in a specific model and generate explanations when making a prediction; *e.g.,* Ripplenet [64], KPRN [70], TMER [18], RuleRec [53], PGPR [77] and KGIN [68] discover meta-paths from knowledge graphs (KGs) as explanations; and (b) post-hoc methods generate explanations after a model makes a prediction, *e.g.,* GraphLime [38], GNNExplainer [82], PGExplainer [49], SubgraphX [85] and GraphMask [59] extract subgraphs and/or features as explanations.

There are several concerns about these methods. (1) These methods extract meta-paths/subgraphs and/or features as explanations, but do not discern what features are decisive and under what conditions the recommendations can be made. (2) The effectiveness of explanations is often measured in terms of (a) fidelity for how faithful explanations are to predictions, as the ratio of GNN predictions that the explanations successfully reproduce, and (b) sparsity for how concise an explanation is [84], as the ratio of the number of selected edges to the number of all edges in a graph. Higher fidelity indicates that more discriminative structures/features are identified, and lower sparsity means that explanations capture mostly important information only. The previous explanation methods often yield fidelity and/or sparsity that do not meet the expectations of practitioners. (3) These methods focus on *local explanations* for a prediction $\mathcal{M}(x, y)$ at specific user x and item y, but do not give *global explanations* to reveal rationales behind the general behavior of $\mathcal{M}$.

Example 1: Consider a fraction of a real-life graph G in Figure 1(a) (the bottom-right shows a simplified version). A GNN model $\mathcal{M}_1$ recommends a movie v "Everything Everywhere All at Once" to a user u "Mike", denoted by $\mathcal{M}_1(u, v)$. As a local explanation for $\mathcal{M}_1(u, v)$, SubgraphX [85] extracts a subgraph from G that includes most of the edges within 2 hops of u and v, as shown in Figure 1(b). However, it does not tell us which features are decisive. GNNExplainer [82] returns both a subgraph and a batch of vertex features, as shown in Figure 1(c). However, the subgraph is disconnected and does not tell the connection between u and v. Besides, the extracted features are the same for all vertices, and do not reveal different impacts of various vertices on the $\mathcal{M}_1$ prediction. □

In light of these, we propose Makex (MAKE senSE), a logic method to explain GNN-based recommendations. Makex introduces a class of logic rules, referred to REPs (Rules for ExPlanations). Given a GNN model $\mathcal{M}$, we discover REPs $Q[x, y](X \rightarrow \mathcal{M}(x, y))$,

*Dandan Lin is the corresponding author.

Proceedings of the VLDB Endowment, Vol. 18, No. 3 ISSN 2150-8097.
doi:10.14778/3712221.3712237

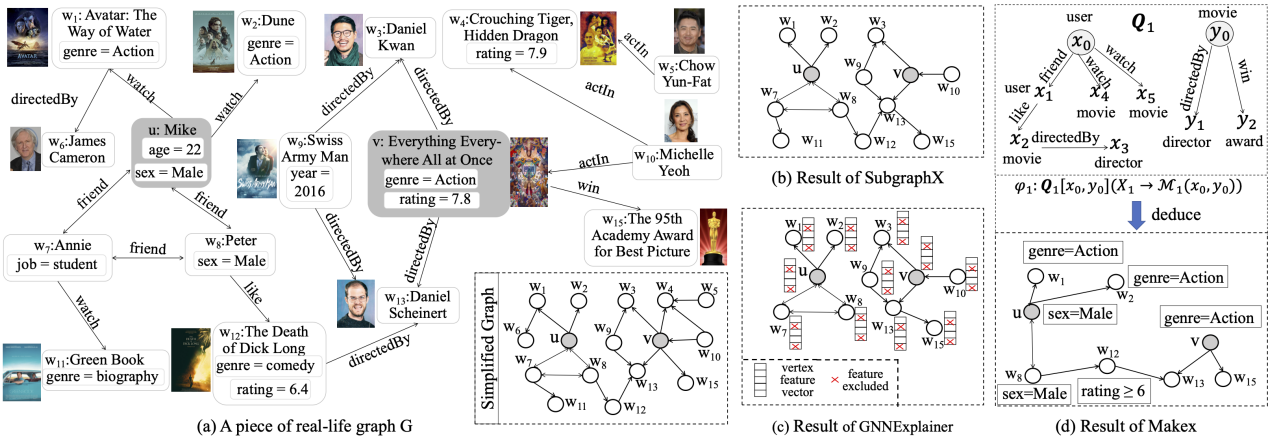


Figure 1: Various local explanations for why model $\mathcal{M}_1$ recommends movie v to user u

where Q is a graph pattern, and X is a collection of predicates. When $\mathcal{M}$ recommends item y to user x, Q identifies topology of x and y relevant to the decision, and X discloses conditions on vertex features. Intuitively, these rules provide *global explanations* to reveal what edges and features are most responsible for $\mathcal{M}$ recommendation. They can also deduce *local explanations* for a prediction $\mathcal{M}(x, y)$ at user x and item y, in terms of satisfied REPs and their *witnesses* (topological matches and vertex features identified by Q and X) as evidence. Moreover, they reveal not only decisive vertex features but also the correlations, interactions and dependencies of the features as conditions under which $\mathcal{M}$ recommends y to x.

Example 2: Continuing with Example 1, we discover REPs for $\mathcal{M}_1$ as its global explanations. As will be seen in Section 6, for a GNN model on a dataset, we can typically discover 110 REPs on average.

We also deduce local explanations with the discovered REPs. An example local explanation is shown in Figure 1(d), with an REP

$$\varphi_1 = Q_1[x_0, y_0](X_1 \rightarrow \mathcal{M}_1(x_0, y_0)),$$

where X_1 is $x_0.\text{sex} = \text{Male} \wedge x_1.\text{sex} = \text{Male} \wedge x_2.\text{rating} \geq 6 \wedge x_3.\text{id} = y_1.\text{id} \wedge x_4.\text{genre} = \text{Action} \wedge x_5.\text{genre} = \text{Action} \wedge y_0.\text{genre} = \text{Action}$. As shown at the top of Figure 1(d), the pattern Q_1 and logic conditions X_1 of φ_1 explain why $\mathcal{M}_1$ recommends a movie y_0 to a male user x_0 *because* (a) x_0 has watched at least two action movies before (which have the same genre as y_0), (b) x_0 has a male friend x_1 who likes high rating movie x_2 (rating $\geq$ 6) directed by x_3, and (c) y_0 is an award-winning movie directed by x_3 (*i.e.,* y_1 since $x_3.\text{id} = y_1.\text{id}$). By finding matches of this REP in the graph, a local explanation for $\mathcal{M}(u, v)$ at user u and movie v can be deduced, as shown at the bottom of Figure 1(d). It concretizes REP φ_1 and says that the action movie v is recommended to Mike because Mike has watched two action movies before, the movie won the 95th academy award for best picture, and it is directed by "Daniel Scheinert", the director of a favorite movie of a male friend Peter of Mike.

Compared to Figures 1(b) and (c), Makex extracts (1) just decisive topology and features, instead of complex subgraphs and one-size-fit-all features; and (2) not only features but also logic conditions under which the ML prediction is made. Moreover, (3) Makex mines REPs in line with the aggregation of neighbor information in GNNs, by means of easy-to-interpret graph patterns, logic conditions and 1-dimensional Weisfeiler-Leman (1-WL) test [72] (see below). □

Contribution & organization. Makex is novel in the following.

(1) REPs*: Rules for explanations* (Section 2). Makex proposes REPs of the form $Q[x, y](X \rightarrow \mathcal{M}(x, y))$. Departing from prior rules, an REP takes a given GNN model $\mathcal{M}(x, y)$ of interest as its *consequence*. The pattern Q and precondition X reveal topology and conditions on vertex features for $\mathcal{M}(x, y)$ to recommend item y to user x, respectively. We define Q as a pair of star patterns to pick most relevant features, such that it is tractable to check matches of Q in a graph. In addition to traditional predicates, X supports the 1-WL test [72] as a predicate. Intuitively, the 1-WL test is a graph-theoretic technique used for comparing the structure of graphs (*e.g.,* to check whether two graphs can be distinguished), and has been widely used in *e.g.,* network analysis and computational chemistry. Since "most existing GNN models for link prediction are based on 1-WL test" [32, 35, 54, 79], the 1-WL test can be used to explain the behaviors of GNN-based recommendations in principle.

(2) Makex*: A system* (Section 3). Given any GNN-based model $\mathcal{M}$ and user-item interaction graph G (enriched with data from knowledge bases), Makex first discovers a set Σ of REPs in G *guided by* $\mathcal{M}$, *i.e.,* given $\mathcal{M}$, it identifies topology Q and precondition X on vertex features for $\mathcal{M}$ to make predictions. The set Σ of REPs reveals the general behaviors of $\mathcal{M}$ and provides global explanations for $\mathcal{M}$. Then whenever $\mathcal{M}$ recommends item v to user u, Makex employs Σ to generate local explanations for prediction $\mathcal{M}(u, v)$ (see below).

(3) Rule discovery (Section 4). We present the algorithm underlying Makex for discovering REPs. As opposed to previous rule discovery algorithm, this algorithm learns REPs that pertain to a given GNN model $\mathcal{M}$. To faithfully simulate $\mathcal{M}$'s predictions, it first finds the patterns of REPs by adapting Monte Carlo tree search (MCTS) [14, 61, 85], and then selects precondition X of decisive predicates under each pattern Q with a *divide-and-conquer approach* on paths.

(4) Top-ranked local explanations (Section 5). When $\mathcal{M}(u, v)$ predicts true in graph G, Makex identifies local explanations for the prediction. It finds REPs of Σ that are applicable to u and v, and their witnesses at u and v. It proposes ranking criteria for REPs and witnesses, and develops a top-k algorithm such that when there are multiple rules applicable and/or a rule has multiple witnesses, it returns top-ranked explanations. The algorithm is in polynomial time (PTIME). It employs pruning strategies for early termination.

(5) Experimental study (Section 6). Using real-life graphs, we empirically find the following. (a) Makex provides effective local explana-

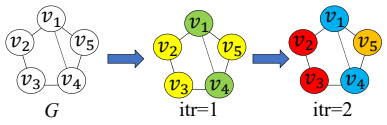


Figure 2: An example of 1-WL test

tions, *e.g.,* the fidelity and sparsity of its top-1 explanation are 0.893 and 0.00225% on average, 80.62% and 3 orders of magnitude better than the baselines, respectively. (b) Makex is efficient in providing local explanations, 75.8X faster than baselines on average, *e.g.,* it takes only 0.38s to generate top-1 explanation on a graph with 119K vertices and 3.7M edges. (c) The global explanations by Makex have higher recognizability and reliability than the baselines by up to 72% and 95%, respectively, *i.e.,* Makex is faithful to GNN predictions. (d) Makex is faster than existing global methods by up to 7X.

We discuss related work in Section 7 and future work in Section 8.

2 RULES FOR EXPLANATIONS

We start with basic notations and review the 1-WL test (Section 2.1). We then present the syntax and semantics of REPs (Section 2.2). Frequently-used notations are summarized in Table 1 (more in [10]).

2.1 Star Patterns and 1-WL Test

Assume two countably infinite sets of symbols, denoted by Λ and Υ, for (vertex and edge) labels and attributes, respectively.

Graphs. We consider directed labeled graphs, specified as $G = (V, E, L, F_A)$, where (a) V is a finite set of vertices; (b) $E \subseteq V \times \Lambda \times V$ is a finite set of edges, in which $e = (v, l, v')$ denotes an edge labeled with $l \in \Lambda$ from vertex v to v'; (c) each vertex $v \in V$ has label $L(v)$ from Λ; and (d) each vertex $v \in V$ carries a tuple $F_A(v) = (A_1 = a_1, \ldots, A_n = a_n)$ of *attributes* of a finite arity, where $A_i \in \Upsilon$ and a_i is a constant, written as $v.A_i = a_i$, and $A_i \neq A_j$ if $i \neq j$, representing features of v. Different vertices may carry different attributes, which are not constrained by a schema like relational databases.

Paths. A *path ρ from a vertex* v_0 in G is a list $\rho = (v_0, v_1, \ldots, v_{n-1}, v_n)$ such that (v_{i-1}, l_{i-1}, v_i) is an edge in G labeled with l_{i-1} ($i \in [1, n]$). We consider *simple* paths on which each vertex appears at most once. The last vertex v_n is called the *leaf* of ρ. A vertex v_i is a *child* of v_{i-1} if there is an edge (v_{i-1}, l_{i-1}, v_i) in E, and v_{i-1} is a *parent* of v_i. The *length* $|\rho|$ *of* ρ is the number n of edges on ρ.

GNN recommendation models. A K-layer GNN-based recommendation model $\mathcal{M}$ consists of three basic parts [29, 34, 41, 66, 68, 74]: (1) a set of user embeddings $\{\mathbf{e}_u^K\}$, (2) a set of item embeddings $\{\mathbf{e}_v^K\}$, both of which are learned by aggregating the information from the K-hop neighbors; and (3) a scoring function $\mathcal{F}$ that takes $\{\mathbf{e}_u^K\}$ and $\{\mathbf{e}_v^K\}$ as input and computes the u-v preference score $\hat{y}_{uv}$. If $\hat{y}_{uv}$ is above a predefined threshold, $\mathcal{M}$ recommends item v to user u.

Pattern matching. We now review star-shaped dual patterns [22].

Star patterns. A *star pattern* is $Q[x_0, \bar{x}] = (V_Q, E_Q, L_Q, \mu)$, where (1) V_Q (resp. E_Q) is a set of pattern vertices (resp. edges) as defined above; (2) L_Q assigns a label of Λ to each vertex in V_Q; (3) $\bar{x}$ is a list of distinct variables, and μ is a bijective mapping from $\bar{x}$ to the vertices of Q; (4) x_0 is a designated variable in $\bar{x}$, referred to as the *center* of Q; and (5) for each $z \in \bar{x}$, there exists a single path from x_0 to z and moreover, z has at most one child, except x_0. For variables $z \in \bar{x}$, we use $\mu(z)$ and z interchangeably if it is clear in the context.

Intuitively, $Q[x_0, \bar{x}]$ has a star shape with center x_0. The center denotes a user/item; it collects properties linked from x_0 via paths.

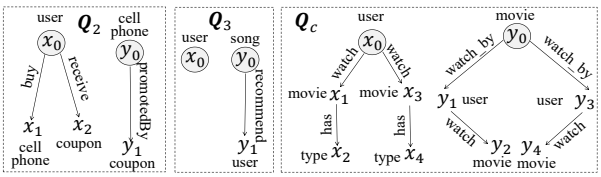


Figure 3: Dual star patterns

Dual patterns. A *dual pattern* is $Q[x_0, y_0] = \langle Q_x[x_0, \bar{x}], Q_y[y_0, \bar{y}] \rangle$, where $Q_x[x_0, \bar{x}]$ and $Q_y[y_0, \bar{y}]$ are disjoint star patterns. *i.e.,* Q_x and Q_y have no common vertices. Intuitively, Q_x (resp. Q_y) represents user x_0 (resp. item y_0), collecting its properties. The use of dual patterns is consistent with GNN models that compute user and item embeddings *separately* before recommendation. Here the stars Q_x and Q_y may have heterogeneous structures in a schemaless graph.

Example 3: As shown in Figure 1(d), dual pattern Q_1 depicts the properties of users and items for making recommendations in Example 1. Note that the centers of star patterns are marked gray. □

Matches. A *match of a star pattern Q in a graph G* is a homomorphic mapping h from the pattern vertices in Q to G such that (a) for each vertex $u \in V_Q$, $L_Q(u) = L(h(u))$, and (b) for each pattern edge (u, l, u') in Q, $(h(u), l, h(u'))$ is an edge in graph G.

A *match of a dual pattern* $Q[x_0, y_0] = \langle Q_x, Q_y \rangle$ in graph G is a homomorphic mapping h from the pattern vertices in $V_{Q_x} \cup V_{Q_y}$ to G.

In the sequel, we refer to a star-shaped dual pattern simply as a *pattern* when it is clear in the context. We refer to a match h of Q as *a match pivoted at* (u, v) if $h(x_0) = u$ and $h(y_0) = v$.

1-WL test. We review 1-dimensional Weisfeiler-Leman (1-WL) test [72]. Given a graph $G = (V, E, L, F_A)$ in which all vertices are initialized with the same color, 1-WL test works by *iterative color refinement* for vertex classification (cf. [31]): for all colors c in the current iteration and all vertices u and v of color c, u and v get different colors in the next iteration if there exists some color d such that u and v have different numbers of neighbors of color d. This refinement iterates until no more changes can be made.

Example 4: Consider the graph in Figure 2 [88]. Initially, all vertices are marked white. Then the coloring is refined iteratively. In the first iteration, v_1 and v_5 are marked with different colors since they have 3 and 2 white children, respectively. After two iterations, no more changes can be made and the final coloring is shown, where v_2 and v_3 have the same color and are put into the same class. □

Properties. The following have been established about 1-WL test.

(1) The 1-WL test takes at most $O((|V| + |E|)\log|V|)$ time (cf. [31]).

(2) GNNs are at most as powerful as the 1-WL test in distinguishing graph structures [31, 32, 54, 79]. Moreover, most GNN recommendation models are based on 1-WL [35], which compute link prediction scores by aggregating pairwise node representations. Hence 1-WL can explain the behaviors of those GNN recommenders in principle.

2.2 REPs: Syntax and Semantics

We next define rules for explaining GNN-based recommendations.

Explaining rules. We start with predicates of the rules.

Predicates. We define a *logic predicate* of a dual pattern $Q[x_0, y_0] = \langle Q_x[x_0, \bar{x}], Q_y[y_0, \bar{y}] \rangle$ in one of the following forms:

$$p ::= x.A \oplus y.B \mid z.A \oplus c \mid 1\text{WL}(x, y_0) \mid 1\text{WL}(x_0, y),$$

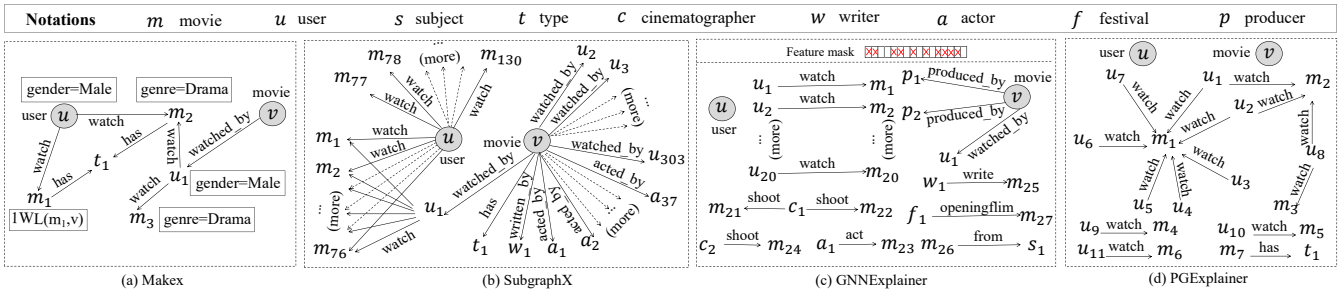


Figure 4: Case study

where $\oplus$ is one of $=, \neq, <, \leq, >, \geq$; $x \in \bar{x}$ and $y \in \bar{y}$ are variables in Q_x and Q_y, respectively, and variable $z \in \bar{x} \cup \bar{y}$; c is a constant; A and B are attributes in Υ. We refer to $x.A \oplus y.B$ and $z.A \oplus c$ as *variable predicate* and *constant predicate* of Q, respectively. We refer to $\mathsf{1WL}(x, y_0)$ as *1-WL predicate*, which predicts true for the existence of an edge (x_0, l, y_0) if x and y_0 are in the same class by 1-WL test and (x_0, l, x) is an edge in Q_x. Intuitively, if a GNN recommends x to x_0 and if y_0 and x are characterized "the same", then the model should recommend y_0 to x_0 as well; similarly for $\mathsf{1WL}(x_0, y)$.

REPs. A *Rule φ for ExPlaining* GNN-based model $\mathcal{M}$ is defined as:

$$Q[x_0, y_0](X \to \mathcal{M}(x_0, y_0)),$$

where $Q[x_0, y_0]$ is a dual pattern, X is a conjunction of logic predicates of $Q[x_0, y_0]$, and $\mathcal{M}$ is a GNN model. Here x and y in variable predicate $x.A \oplus y.B$ are the leaf/center of stars Q_x and Q_y in Q, respectively; each vertex carries at most one such predicate. We refer to Q and $X \to \mathcal{M}(x_0, y_0)$ as the *pattern* and *dependency*, and X and $\mathcal{M}(x_0, y_0)$ as the *precondition* and *consequence* of φ, respectively.

One can plug in an arbitrary GNN-based recommendation model $\mathcal{M}$, *e.g.,* PinSAGE [81], HGT [36] and KGAT [66]. Here $\mathcal{M}(x_0, y_0)$ is true if and only if $\mathcal{M}$ predicts that the strength of how user x_0 likes item y_0 is above a predefined threshold.

Intuitively, (a) pattern $Q[x_0, y_0]$ finds relevant sub-structures of user x_0 and item y_0 inspected by $\mathcal{M}$, and each variable z in Q may carry various attributes (features); and (b) precondition X catches the interactions and dependencies of features. Together Q and X explain why $\mathcal{M}$ suggests y_0 to x_0. In particular, $x.A \oplus y.B$ compares features between the associated leaves or pivots, and 1WL predicates check whether there exists an edge between x_0 and y_0 by utilizing the classification of variables in Q by the 1-WL test.

We consider REPs that take the same recommendation model $\mathcal{M}$ as their consequence, referred to as REPs *pertaining to model* $\mathcal{M}$.

Example 5: Below are example REPs with patterns in Figure 3.

(1) An REP φ_1 is given in Example 2 to explain why model $\mathcal{M}_1$ recommends movie y_0 to user x_0. It extracts the important features of x_0 and y_0 from the movie-watching history and social links of x_0.

(2) $\varphi_2 = Q_2[x_0, y_0](X_2 \to \mathcal{M}_2(x_0, y_0))$, where X_2 is x_0.occupation = College Student $\wedge x_1$.brand = y_0.brand $\wedge x_2$.id = y_1.id $\wedge y_1$.type = Education Pricing. Here φ_2 says that if x_0 is a college student, she has bought a phone x_1 before and received a coupon that can be used to buy y_0, a cell phone of the same brand promoted with education pricing [8], then $\mathcal{M}_2$ suggests cell phone y_0 to user x_0.

(3) $\varphi_3 = Q_3[x_0, y_0](\mathsf{1WL}(x_0, y_1) \wedge X_3 \to \mathcal{M}(x_0, y_0))$, where X_3 is x_0.occupation = Classical Musician $\wedge y_1$.occupation = Classical Musician. It tells that $\mathcal{M}$ suggests song y_0 to user x_0 because (a) both x_0 and y_1 are classical musicians (by X_3), and (b) x_0 and y_1 are in the same class by 1WL and y_0 was recommended to y_1 before.

(4) An REP to explain KGAT is $\varphi_c = Q_c[x_0, y_0](X_c \to \mathcal{M}_c(x_0, y_0))$, where Q_c is shown in Figure 3 and X_c is x_0.gender = Male $\wedge$ $\mathsf{1WL}(x_1, y_0) \wedge x_3$.genre = Drama $\wedge y_1$.gender = Male $\wedge y_2$.genre = Drama $\wedge y_3$.gender = Male $\wedge y_4$.genre = Drama. It says that movie y_0 is recommended to a male x_0 by $\mathcal{M}_c$ because (a) x_0 has watched a Drama movie x_3 before that has the type x_4, (b) x_0 has also watched a movie x_1 that has the same class as movie y_0 predicted by 1WL, (c) y_0 has been watched by two males y_1 and y_3, and both of them also watched Drama movies y_2 and y_4. Intuitively, x_0 has the same preference as those who watched movie y_0, y_0 has the same type as a movie watched by x_0, and thus, y_0 is recommended to x_0.

A match of Q_c is shown in Figure 4(a); it highlights the subgraph (specifying, *e.g.,* the watching history of x_0) and important features (*e.g.,* the genre, gender and 1WL information). This witness reproduces the KGAT recommendation of a movie v "Bird on a Wire" to a user u (ID 7989) on MovieLens [33]; *i.e.,* KGAT on the subgraph yields the same prediction as on the entire MovieLens graph.

In comparison, although SubgraphX also reproduces the prediction $\mathcal{M}_c(u, v)$, its explanation subgraph (Figure 4(b)) is denser than the one identified by Makex since it includes most edges within 2 hops of u and v, *e.g.,* 76 edges from user u_1 to a movie that was also watched by user u, and 303 edges from movie v to a user. GNNExplainer (Figure 4(c)) provides a disconnected subgraph that includes only three edges from the perspective of movie v but many edges unrelated to either user u or movie v. Worse still, all vertices in the explanation share the same mask on feature vectors and this explanation fails to reproduce the prediction $\mathcal{M}_c(u, v)$. PGExplainer (Figure 4(d)) cannot reproduce the prediction either. Its explanation not only misses important edges from the perspective of user u or movie v, but also includes noisy edges, *e.g.,* (u_1, watch, m_1), since it learns edge importance for all training pairs at once. We conducted a user study by using this case (see Section 6 for more details). □

Justification. REPs are defined to strike a balance between the expressivity and complexity for explaining GNN recommendations.

(1) The reason for adopting dual star patterns is twofold. (a) Dual patterns capture the rationales behind GNN recommendation models. They collect various neighboring information that users/items receive for aggregation, by exploiting *multiple paths*, similar to *meta-paths* that are widely-used in GNN models [36, 41, 66, 68]. Besides, we can learn different sub-structures centered at users and items, as GNN recommendation models compute user and item embeddings *separately*. Thus, dual star patterns in REPs identify decisive topology only, and make explanations faithful and sparse. As will be seen in Section 6, REPs with such patterns are able to reproduce GNN predictions. (b) It is in PTIME (polynomial time) to check

Table 1: Notations

Notations	Definitions
G, $Q[x_0, y_0]$	graph, dual star pattern $Q[x_0, y_0] = \langle Q_x[x_0, \bar{x}], Q_y[y_0, \bar{y}]\rangle$
$\mathcal{M}(x_0, y_0)$	a GNN model that predicts how user x_0 likes item y_0
1WL(x, y)	the predicate for 1-WL test
φ, Σ	REP $\varphi = Q[x_0, y_0](X \to \mathcal{M}(x_0, y_0))$, a set of REPs
ρ	a path $\rho = (z_0, z_1, \ldots, z_n)$ with length $\|\rho\| = n$
h	a match of Q; a witness of φ if $h \models \varphi$
(φ, h)	an evidence (an applicable φ and a witness of φ for $\mathcal{M}(u, v)$)
$\mathbb{H}_{(u,v)}(\Sigma, G)$	the set of evidences for $\mathcal{M}(u, v)$ by REPs in Σ.
$s(\varphi, h)$	the ranking/importance score of (φ, h)
$s(w)$, $\hat{s}(z)$	the importance score of w, upper bound of all matches of z
Ψ, T_k	a heap of top-k evidences, the k-th highest ranking score
$\hat{s}(\varphi)$	a score upper bound for all possible witnesses h of φ
$C_\rho(z), C_\varphi(z)$	a set of vertices w in G s.t. there exists h of ρ/φ and $h(z) = w$

the existence of matches of such a pattern [22]. In contrast, pattern matching is NP-complete for generic graph patterns (cf. [30]).

(2) We define REPs such that it is in PTIME to compute explanations with REPs (see Section 5). Moreover, since 1WL is at least as powerful as most GNN-based recommendation models $\mathcal{M}$, REPs can explain different predictions of such $\mathcal{M}$ in principle.

(3) Departing from prior methods, *e.g.,* SubgraphX, GNNExplainer and PGExplainer, Makex starts from REPs as global explanations and deduces local explanations; it develops very different methods to mine patterns, learn dependencies and rank explanations (see Sections 3-5). As will be seen in Section 6, even if we equipped GNNExplaier with REPs by replacing its input with a smaller subgraph deduced from REPs, Makex still beats it by up to 11.50% since GNNExplainer cannot find decisive features for different vertices.

Semantics. Denote by h a match of the pattern Q of a REP $\varphi = Q[x_0, y_0](X \to \mathcal{M}(x_0, y_0))$ in G, and by p a predicate of X. Match h *satisfies* p, denoted by $h \models p$, if the following conditions are satisfied: (a) when p is $x.A \oplus y.B$, the vertex $h(x)$ (resp. $h(y)$) carries attribute A (resp. B), and $h(x).A \oplus h(y).B$; similarly for $z.A \oplus c$, (b) when p is 1WL(x, y_0) (resp. 1WL(x_0, y)), the 1-WL test predicts that $h(y_0)$ (resp. $h(x_0)$) is in the same class as $h(x)$ that was already linked to $h(x_0)$ (resp. $h(y_0)$); and (c) for $\mathcal{M}(x_0, y_0)$, $\mathcal{M}$ predicts true at $(h(x_0), h(y_0))$, *i.e.,* it suggests to recommend $h(y_0)$ to $h(x_0)$.

For $\varphi = Q[x_0, y_0](X \to \mathcal{M}(x_0, y_0))$, we write $h \models X$ if h satisfies *all* predicates in X. We write $h \models \varphi$ if $h \models X$ entails $h \models \mathcal{M}(x_0, y_0)$.

A graph G *satisfies* $\varphi = Q[x_0, y_0](X \to \mathcal{M}(x_0, y_0))$, denoted by $G \models \varphi$, if for all matches h of $Q[x_0, y_0]$ in G such that $h \models X$, $h \models \varphi$. We write $G \models \Sigma$ for a set Σ of REPs if for all $\varphi \in \Sigma$, $G \models \varphi$.

For a pair (u, v) of user and item, we refer to mapping h as a *witness of* φ *at* (u, v) if $h \models \varphi$, $h(x_0) = u$ and $h(y_0) = v$. We say that REP φ is *applicable at* (u, v) if there exists a witness at (u, v).

3 MAKEX: AN EXPLANATION SYSTEM

This section presents an overview of Makex for generating logic explanations for recommendations of GNN-based models $\mathcal{M}$.

To simplify the discussion, we focus on CTR (click-through rate), *i.e.,* $\mathcal{M}(x, y)$ recommends item y to user x if the strength of its prediction is above a predefined threshold. This said, our method can also be adapted for top-k recommendation, which suggests to each user x at most k items y that have top ranked $\mathcal{M}(x, y)$ strengths.

Explanations. We first formalize the notions of global and local explanations. Consider a GNN recommendation model $\mathcal{M}$ and a user-item interaction graph G (enriched with knowledge graphs).

Global explanations. Given $\mathcal{M}$ and a graph G, Makex discovers a set Σ of REPs to explain the predictions of $\mathcal{M}$ on G (see below). For

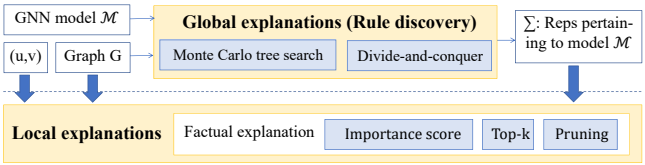


Figure 5: The workflow of Makex

GNN models such as PinSAGE [81], HGT [36] and KGAT [66] tested in our experiments, the set Σ includes at most 172 REPs in all real-life datasets adopted (see Section 6). These REPs determine what features are most responsible for $\mathcal{M}$'s predictions, characterize the general behaviors of model $\mathcal{M}$, cover different cases of $\mathcal{M}$'s predictions via the 1-WL test, and can serve as global explanations of $\mathcal{M}$.

Local explanations. Denote by $\mathbb{H}_{(u,v)}(\Sigma, G)$ the set of pairs (φ, h) for all REPs $\varphi \in \Sigma$ applicable at (u, v) and all witnesses h of φ at (u, v) in G, where each pair (φ, h) is referred to as an *evidence* for the prediction $\mathcal{M}(u, v)$. For a pair (u, v) of user and item in graph G and a set Σ of REPs for model $\mathcal{M}$, the *local explanations* for $\mathcal{M}$ to recommend item v to user u are simply defined as $\mathbb{H}_{(u,v)}(\Sigma, G)$.

Here each evidence $(\varphi, h) \in \mathbb{H}_{(u,v)}(\Sigma, G)$ is a sufficient condition for $\mathcal{M}$ to predict true at u and v. Let $\varphi = Q(x_0, y_0)(X \to \mathcal{M}(x_0, y_0))$. Then h exhibits how φ is enforced and says that $\mathcal{M}(u, v)$ is true because $h \models X$, providing factual explanation (see Example 5).

Modules. As shown in Figure 5, given $\mathcal{M}$ and G as described above, Makex first learns a set Σ of REPs pertaining to model $\mathcal{M}$ as global explanations. Then whenever $\mathcal{M}$ suggests to recommend item v to user u, Makex computes local explanations for prediction $\mathcal{M}(u, v)$ upon request. It consists of the following main modules.

(1) Rule discovery (Section 4). Given graph G and model $\mathcal{M}$, it discovers a set Σ of REPs pertaining to $\mathcal{M}$ to capture features/patterns for $\mathcal{M}$ to make predictions. The discovery is conducted *once offline*, *i.e.,* we re-use the set Σ of REPs for each input user-item pair (u, v). The discovery is *guided by* $\mathcal{M}$ and thus, the REPs in Σ fit $\mathcal{M}$ well.

(2) Local explanations (Section 5). Given the set Σ and a pair (u, v), Makex computes the set $\mathbb{H}_{(u,v)}(\Sigma, G)$ as local explanations for $\mathcal{M}$ to recommend v to u. Makex ranks the pairs (φ, h) with *importance scores* such that users are provided with top-ranked pairs at a time, by developing a top-k algorithm with effective pruning strategies.

4 MODEL-GUIDED RULE DISCOVERY

This section develops an algorithm, RepsLearner, to discover REPs that faithfully explain a given GNN model, unlike prior rule miners.

Criteria. We start with two measures to evaluate the quality of REPs. Consider REP $\varphi = Q[x_0, y_0](X \to \mathcal{M}(x_0, y_0))$ and graph G.

Support. This is to measure how often an REP φ can be applied in graph G. More specifically, the *support* of φ in G is defined as:

$$\text{supp}(\varphi, G) = \|Q(x_0, y_0, G, X \wedge \mathcal{M}(x_0, y_0))\|.$$

Here $Q(x_0, y_0, G, X \wedge \mathcal{M}(x_0, y_0))$ is the set of pairs $(h(x_0), h(y_0))$ for all matches h of Q in G such that $h \models X \wedge \mathcal{M}(x_0, y_0)$. Intuitively, the higher $\text{supp}(\varphi, G)$ is, the more frequent φ can be applied to G.

Confidence. It measures how strong the connection between the precondition X and prediction $\mathcal{M}(x_0, y_0)$ is. The confidence of φ in G is:

$$\text{conf}(\varphi, G) = \frac{\text{supp}(\varphi, G)}{\|Q(x_0, y_0, G, X)\|}.$$

REP discovery. We formalize the discovery problem as follows.

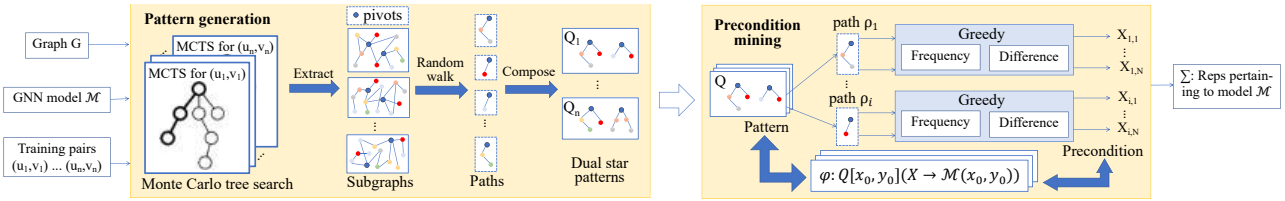


Figure 6: Overview of RepsLearner

- *Input*: A graph G, a GNN model $\mathcal{M}$, a support threshold $\sigma > 0$, a confidence threshold $\delta > 0$, and positive integers α_1 and α_2.
- *Output*: A set Σ of REPs pertaining to $\mathcal{M}$ such that for each $\varphi \in \Sigma$, we have $\text{supp}(\varphi, G) \geq \sigma$, $\text{conf}(\varphi, G) \geq \delta$, and Q has at most α_1 paths where each path has at most α_2 variables.

Here α_1 and α_2 are mostly to control the cost of rule discovery.

Overview. As outlined in Figure 6, RepsLearner learns REPs as global explanations for a given GNN model $\mathcal{M}$ mainly in two step.

(1) Pattern generation: It first generates dual patterns $Q[x_0, y_0] = \langle Q_x, Q_y \rangle$ pertaining to $\mathcal{M}$. To do this, (a) it computes a subgraph for each user-item pair (u, v), by adapting *Monte Carlo tree search (MCTS)*. (b) Then it uses *random walks* starting from user u (resp. item v) in the subgraph to get a list of paths; these paths compose star patterns Q_x (resp. Q_y) centered at x_0 (resp. y_0).

(2) Precondition mining: It then mines a set of *frequent* (*i.e.,* high support) and *diverse* preconditions X using a divide-and-conquer approach and pairs each X with $Q[x_0, y_0]$ to form final REPs.

Below we present the details of the two steps of RepsLearner.

(1) Pattern generation. The adapted MCTS and random walks work to get a set of star patterns retaining the predictions of $\mathcal{M}$ as follows.

(a) Monte Carlo tree search (MCTS). For each pair (u, v) in G, if $\mathcal{M}$ predicts true at (u, v), we build a search tree in which the root is associated with the K-hop neighborhood graph $G_K(u, v)$ of u and v since a K-layer GNN model $\mathcal{M}$ aggregates information from the K-hop neighbors of u and v; each of the other tree nodes is associated with a connected subgraph of $G_K(u, v)$. A MCTS iteration selects a path from the root to a leaf. Then the subgraph $G_{\text{sg}}(u, v)$ at the leaf is evaluated, by comparing the recommendation strengths of $\mathcal{M}(u, v)$ on $G_{\text{sg}}(u, v)$ vs. on $G_K(u, v)$. The most promising subgraph $G_{\text{sg}}(u, v)$ is returned for (u, v) such that $\mathcal{M}$ could *reproduce* its prediction in $G_K(u, v)$, *i.e.,* $\mathcal{M}(u, v)$ predicts the same in $G_{\text{sg}}(u, v)$ as in $G_K(u, v)$.

To speed up, we only build subgraphs for a subset of user-item pairs, and drop those with isomorphic K-hop neighborhoods. As will be observed in Section 6, this strategy can cover enough explanation scenarios so as to achieve good global performance.

(b) Random walks. Since it is time-consuming to get all eligible paths in $G_{\text{sg}}(u, v)$, we simulate random walks from user u in $G_{\text{sg}}(u, v)$, each of which produces a path. To avoid unending walking, we adopt an α-termination probability for simulation. More specifically, a path starts from u and at each step, it either (a) terminates with α probability, or (b) moves to an out-neighbor of the current vertex with $(1-\alpha)$ probability. For each simulated path with at most α_2 steps, we extract a path pattern by replacing the vertices (resp. edges) by pattern vertices (resp. edges) with the same labels. By composing at most α_1 most frequently-visited path patterns for each star pattern Q_x, we obtain a number of star patterns to represent the substructure pivoted at user u; similarly for Q_y at item v. Taking each Q_x and Q_y together, we get a dual pattern $Q[x_0, y_0] = \langle Q_x, Q_y \rangle$.

(2) Precondition mining. For each $Q[x_0, y_0]$, we propose a divide-and-conquer approach to mining preconditions X to form REPs, in three steps. (a) The pattern Q is first divided into independent paths. (b) For each path ρ, we generate a set $\mathcal{X}_\rho$ of N independent candidate preconditions (see below), *i.e.,* $\mathcal{X}_\rho = \{X_{\rho,1}, X_{\rho,2}, ..., X_{\rho,N}\}$, where N is a hyper-parameter and each $X_{\rho,i}$ is a precondition ($i \in [1, N]$), *i.e.,* a conjunction of predicates, on path ρ. (c) We compose the paths of Q, where each path ρ is associated with a candidate X_ρ in $\mathcal{X}_\rho$, into candidate REPs $\varphi = Q[x_0, y_0](\wedge_{\rho \in Q} X_\rho \to \mathcal{M}(x_0, y_0))$ and select those that are above the support/confidence thresholds.

Generating candidate preconditions. Given a path $\rho = (z_0, z_1, \ldots, z_n)$, the preconditions in $\mathcal{X}_\rho$ are generated one by one, until N candidate preconditions are in place. We show how the i-th precondition $X_{\rho,i}$ in $\mathcal{X}_\rho$ is generated after we get $i-1$ preconditions in $\mathcal{X}_\rho$. We first initialize $X_{\rho,i}$ as empty. Then we traverse the path ρ in rounds, starting from the center z_0 to the leaf z_n, to select the promising predicates defined on each variable. Specifically, at the j-th round ($j \in [0, n]$), we process predicates defined on variable z_j and add promising predicates to $X_{\rho,i}$ by using a greedy strategy (see below).

To decide which predicates p should be added to $X_{\rho,i}$, we consider both the support of $X_{\rho,i} \wedge p$ (*i.e.,* whether the resulting precondition can be frequently applied in G) and its difference compared with those already in $\mathcal{X}_\rho$. To measure this difference, we define $\text{diff}(X_{\rho,i} \wedge p, \mathcal{X}_\rho, G)$ to be the number of pairs $(h(x_0), h(y_0))$ such that h satisfies $X_{\rho,i} \wedge p$ but not any of the first $i-1$ preconditions in $\mathcal{X}_\rho$. Then we define an indicator score for p on $X_{\rho,i}$ to be:

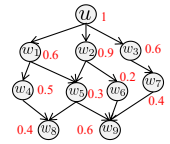
$$I(X_{\rho,i}, p) = w_s \cdot \|Q(x_0, y_0, G, X_{\rho,i} \wedge p)\| + w_d \cdot \text{diff}(X_{\rho,i} \wedge p, \mathcal{X}_\rho, G),$$

where w_s and w_d are weights such that $w_s + w_d = 1$. Intuitively, the predicates with high indicator scores are preferred and added to $X_{\rho,i}$. As will be seen in Section 6, the larger the value of N, the better the performance of the discovered rules as global explanations.

1WL *predicates.* We pre-compute the classes of all vertices in G by the 1-WL test and treat the class as a constant attribute for a vertex. If two vertices x and y_0 have the same class and if x is recommended to x_0, we conclude that $\text{1WL}(x, y_0)$ holds; similarly for $\text{1WL}(x_0, y)$.

Example 6: We show how RepsLearner finds φ_1 in Figure 1. After generating the pattern Q_1 by MCTS and random walks, Q_1 is first divided into multiple paths, *e.g.,* (x_0, x_1, x_2, x_3) and (x_0, x_4). Consider $\rho = (x_0, x_4)$, $N = 2$ and assume that the first precondition in $\mathcal{X}_\rho$ is $X_a = \{x_0.\text{sex} = \text{Male}, x_4.\text{genre} = \text{Action}\}$. We prefer diverse preconditions, *e.g.,* $X_b = \{x_0.\text{sex} = \text{Female}, x_4.\text{genre} = \text{Romance}\}$ instead of $X_c = \{x_0.\text{sex} = \text{Male}, x_4.\text{genre} = \text{Thriller}\}$, so that more matches can find satisfiable preconditions in $\mathcal{X}_\rho$; similarly for other paths. By composing the results of all paths of Q_1, we can get φ_1. □

Complexity. Denote by (a) c_{mcts} the unit cost for computing a subgraph via MCTS for a given training pair (see [14, 61] for more); (b) $|R|$ the number of random walks simulated on a given subgraph; and (c) $|\mathcal{T}|$ the number of training pairs used. Then RepsLearner takes $O(c_{\text{mcts}}|\mathcal{T}| + |R||\mathcal{T}|)$ time to generate $O(|\mathcal{T}|)$ patterns. Besides, for each pattern Q, RepsLearner generates



(a) A subgraph pivoted at u

Q_x	$z \in \rho$	$C_{\varphi_1}(z)$	$\hat{s}(z)$
ρ	x_0	u	1
	x_1	w_1, w_2	0.9
	x_2	w_4, w_5, w_6	0.5
	x_3	w_8, w_9	0.6

(b) The vertex set C_{φ_1} for φ_1

z	$h(z)$	Δ_w
x_1	w_1	0
	w_2	0.3
x_3	w_8	0
	w_9	0.2

z	$h(z)$	Δ_w
x_2	w_4	0.2
	w_5	0
	w_6	0

(c) Indicator score Δ_w

Q_x	$z \in \rho$	$C_{\varphi_1}(z)$
ρ	x_0	u
	x_1	w_1
	x_2	w_4, w_5
	x_3	w_8, w_9

(d) C_{φ_1} after removing w_2

Figure 7: Algorithmic examples

$O(N^{\alpha_1})$ preconditions to form candidate REPs, since Q has at most α_1 paths and each path has N candidate preconditions, where each precondition of a path can be generated in $O(\alpha_2 c_z)$ time, where c_z is the unit cost for selecting predicates for a variable z, resulting in $O(c_{\text{mcts}}|\mathcal{T}| + |R||\mathcal{T}| + \alpha_1\alpha_2 c_z N|\mathcal{T}| + N^{\alpha_1}|\mathcal{T}|)$ total time.

5 TOP-RANKED LOCAL EXPLANATIONS

This section develops an algorithm to generate evidences as local explanations for GNN recommendations. The problem is as follows.

- *Input*: Graph G, a set Σ of REPs pertaining to model $\mathcal{M}$, and a user-item pair (u, v) in G with $\mathcal{M}(u, v) = \text{true}$.
- *Output*: The set $\mathbb{H}_{(u,v)}(\Sigma, G)$ of evidences for $\mathcal{M}(u, v)$; each evidence is a pair (φ, h) for $\varphi \in \Sigma$ and a witness h of φ at (u, v).

A PTIME algorithm. An algorithm for computing $\mathbb{H}_{(u,v)}(\Sigma, G)$ works as follows: for each REP $\varphi = Q[x_0, y_0](X \to \mathcal{M}(x_0, y_0))$ in Σ, find all matches h of Q in G at (u, v) and check whether $h \models \varphi$; if so, add (φ, h) to $\mathbb{H}_{(u,v)}(\Sigma, G)$. Here h is computed by assembling the witnesses of paths in each star of Q at its center, and a witness of a path is inductively defined on its vertices via scattered matches (see Section 5.3). Moreover, by the definition of REPs, we verify that it is in PTIME to check whether $h \models \varphi$ (see Theorem 1).

However, this algorithm may not work very well in practice. (a) When G is dense, it is still costly to find all witnesses of φ at (u, v) although it is in PTIME. (b) There are possibly multiple REPs applicable at (u, v) and multiple witnesses of φ at (u, v) for each φ. The users may not want the enumeration of all these. Instead, they want "top-ranked" evidences so that each evidence serves as an explanation for the prediction at (u, v). In light of these, we develop a top-k algorithm for computing top-ranked evidences.

5.1 Ranking Score

Since each evidence (φ, h) in $\mathbb{H}_{(u,v)}(\Sigma, G)$ consists of an applicable REP φ and a witness h of φ, we define the ranking score of an evidence (φ, h), denoted by $s(\varphi, h)$, by taking both its rule importance and witness importance into account. Formally, we define:

$$s(\varphi, h) = s(\varphi) \cdot s_\varphi(h),$$

where $s(\varphi)$ is the rule importance for measuring the amount of information expressed in φ, and $s_\varphi(h)$ is the witness importance for characterizing the strength of h for the prediction of $\mathcal{M}(u, v)$. The higher the score, the more important the rule/witness (see below).

Intuition. Below we present the intuition of rule/witness importance. Interested readers can find the detailed definitions in [10].

(a) Rule importance. For $\varphi = Q[x_0, y_0](X \to \mathcal{M}(x_0, y_0))$, define

$$s(\varphi) = s_\varphi(Q) \cdot s_\varphi(X),$$

where $s_\varphi(Q)$ and $s_\varphi(X)$ are the importance of pattern Q and precondition X, respectively. For the pattern importance, we consider both the number of paths and the length of the paths to prioritize succinct pattern, while for precondition importance, we adopt the GINI value as an effective criterion for measuring how well X divides witnesses into different classes according to the predictions of $\mathcal{M}$.

Input: A graph G, a set Σ of REPs, a pair (u, v), and an integer k.
Output: A heap Ψ of top-k evidences in G at (u, v).
1. sort REPs φ in Σ in decreasing order based on $s(\varphi)$; $\Psi := \emptyset$;
2. **for** each REP $\varphi = Q[x_0, y_0](X \to \mathcal{M}(x_0, y_0))$ in Σ **do**
3. compute C_φ for φ; $\hat{s}(z) = \max_{w \in C_\varphi(z)} s(w)$;
4. T_k := the k-th highest score in Ψ; $\hat{s}(\varphi) := s(\varphi) \cdot \big(\sum_{z \in \bar{x} \cup \bar{y}} \hat{s}(z)\big)$;
5. **while** $\hat{s}(\varphi) > T_k$ **do**
6. $(w, z) :=$ SelectVertexUB(C_φ);
7. **while** $h =$ NextMatch(w, G, φ) and $h \neq \emptyset$ **do**
8. **if** $s(\varphi, h) > T_k$ **then** update Ψ and T_k by (φ, h);
9. $C_\varphi(z) := C_\varphi(z) \setminus \{w\}$; refine C_φ; compute (tighter) $\hat{s}(\varphi)$;
10. **return** Ψ;

Figure 8: Top-k algorithm TopkEx

(b) Witness importance. Even for the same φ, its witnesses are not equally potent for recommendation. Suppose φ suggests movie y to user x if x has watched similar movie z in the graph and z.rating ≥ 6. Consider two witnesses h_a and h_b of φ, where $h_a(z)$ and $h_b(z)$ map to "Titanic" [2] and "A Walk in the Clouds" [1], respectively. Although both satisfy z.rating ≥ 6, h_a is more promising since (a) the actual rating of $h_a(z) = 7.9$ is higher than $h_b(z) = 6.7$; (b) $h_a(z)$ was rated by 1.3M users, as opposed to 36K of $h_b(z)$; thus $h_a(z)$ is a hub vertex which will contribute more to the final recommendation.

Motivated by this, we quantify the witness importance $s_\varphi(h)$ for each h, by summing up the scores of each vertex in h, *i.e.,*

$$s_\varphi(h) = \sum_{z \in \bar{x} \cup \bar{y}} s(h(z)),$$

where $s(h(z))$ is the score of $h(z)$ and it is defined by considering both actual values and degrees of vertices. Intuitively, we assign a larger score to a vertex with a larger degree (or with a actual value closer to the "optimal" value, *e.g.,* a higher rating is more desirable).

Summary. Taken together, the score of (φ, h) can be written as:

$$s(\varphi, h) = s(\varphi) \cdot s_\varphi(h) = s(\varphi) \cdot \big(\sum_{z \in \bar{x} \cup \bar{y}} s(h(z))\big).$$

Example 7: Consider φ_1 in Example 5. Take a path $\rho_x = (x_0, x_1, x_2, x_3)$ of Q_1 and its match $h_1(\rho_x) = (u, w_2, w_5, w_8)$ in Figure 7(a) as an example, where $s(w)$ of each $w = h(x)$ is colored in red. Then $\sum_{w \in h_1(\rho_x)} s(w) = 1 + 0.9 + 0.3 + 0.4 = 2.6$. Suppose after summing up the importance scores for all matching vertices in $h_1(\varphi_1)$, $s_{\varphi_1}(h_1)$ is 3.2. If the rule importance of φ_1 is 0.9, then the ranking score of evidence (φ_1, h_1) is $s(\varphi_1, h_1) = s(\varphi_1) \cdot s_{\varphi_1}(h_1) = 2.9$. □

Problem. Based on the ranking score, we define the top-k problem.

- *Input*: G, Σ, (u, v) as above, and a positive integer k.
- *Output*: A set Ψ of k evidences s.t. each $(\varphi, h) \in \Psi$ is in $\mathbb{H}_{(u,v)}(\Sigma, G)$, and $s(\varphi, h) \geq s(\varphi', h')$ for all $(\varphi', h') \in \mathbb{H}_{(u,v)}(\Sigma, G) \setminus \Psi$.

5.2 Pruning with Score Upper Bound

Instead of computing the scores of every evidence, we utilize a *pruning strategy* to filter out those evidences unlikely to the top-k.

Pruning strategy. Given an REP φ, the core of our pruning strategy is a *score upper bound*, denoted by $\hat{s}(\varphi)$, of the ranking scores of *all* evidences involving φ, *i.e.,* $\hat{s}(\varphi) \geq s(\varphi, h)$ where h is an *arbitrary* witness of φ. If $\hat{s}(\varphi)$ is small, all witnesses of φ cannot constitute high-ranked evidences and thus, they may be pruned early.

Formally, we compute the score upper bound $\hat{s}(\varphi)$ as follows:

$$\hat{s}(\varphi) = s(\varphi) \cdot \big(\sum_{z \in \bar{x} \cup \bar{y}} \hat{s}(z)\big), \quad (1)$$

where $\hat{s}(z)$ is the score upper bound of all vertices in G that can

be mapped to a variable $z \in \bar{x} \cup \bar{y}$, *i.e.,* $\hat{s}(z) = \max_{\forall h \text{ of } \varphi} s(h(z))$. One may want to compute $\hat{s}(z)$ by enumerating all witnesses h of φ in G for the maximum $s(h(z))$. However, it is costly to explicitly enumerate all witnesses. Below we propose a more efficient method.

Score upper bounds. Given an REP φ and pair (u, v), we compute its witnesses pivoted at (u, v) by decomposing Q into sets of disjoint paths and validating their witnesses independently. The complete witnesses of φ are obtained by combining the results for all paths.

Denote by ρ the path from center x_0 (resp. y_0) to a leaf of Q_x (resp. Q_y). We compute the *witnesses* of $\rho = (z_0, \ldots, z_n)$ pivoted at u (resp. v), by adopting a notion of *scattered matches* of each $z_i \in \rho$.

(1) Scattered matches. For $z_i \in \rho$ ($i \in [0, n]$), the *scattered matches* of z_i in G include all vertices w_i of G satisfying *refinement conditions*: (a) $L(w_i) = L_Q(z_i)$, (b) w_i satisfies all constant predicates and 1-WL predicates on z_i in X, and (c) when $i < n$, there is an edge (w_i, l, w_{i+1}) in G such that w_{i+1} is a scattered match of z_{i+1}.

Similarly, we define the following. For $z_i \in \rho$ ($i \in [0, n]$), the *inverse scattered matches* of z_i in G include all vertices w_i in G such that refinement conditions (a) and (b) are satisfied as above, and (c) when $i > 0$, there is an edge (w_{i-1}, l, w_i) in G such that w_{i-1} is a scattered match of z_{i-1}. The *bidirectional scattered matches* of z_i contain vertices w_i such that refinement conditions (a) and (b) are satisfied as above and (c) if z_i has a child (resp. parent) in ρ, then there exists an edge (w_i, l, w_{i+1}) (resp. (w_{i-1}, l, w_i)) in G such that w_{i+1} (resp. w_{i-1}) is a bidirectional scattered match of z_{i+1} (resp. z_{i-1}).

Note that when there is a variable predicate defined across two paths ρ_x and ρ_y, the two paths are processed together and we consider an additional refinement condition, *i.e.,* whether the variable predicate is satisfied, when computing the bidirectional scattered matches. Otherwise, each path is processed independently.

(2) PTIME *process.* Denote by $C_\rho(z)$ (resp. C_ρ) the set of bidirectional scattered matches of z (resp. all variables) in ρ. We compute C_ρ by iteratively checking the refinement conditions of (inverse) scattered matches of variables from ρ in G by *dynamic programming*.

Intuitively, a complete match of ρ via homomorphism maps the entire set of variables from ρ as a whole. In contrast, the scattered matches are defined on distinct variables from ρ, while they recursively enforce (partial) requirements on edge connections as in standard pattern matching, reducing (possibly) exponential complete matches to $O(|G||\rho|)$ scattered matches. Moreover, since the satisfaction of predicates in X is also enforced, the matches are witnesses of ρ. By induction on the path and the semantics of scattered matches, one can verify that every bidirectional scattered matches of z in $C_\rho(z)$ must appear in complete witnesses of ρ. Indeed, we can assemble $C_\rho(z)$ of different $z \in \rho$ to restore complete witnesses of ρ.

Denote by C_φ the union set of C_ρ for all paths ρ in φ. Once $C_\varphi(z)$ is computed where $C_\varphi(z) = C_\rho(z)$ for $z \in \rho$, the score upper bound $\hat{s}(z)$ of the vertices in G that are mapped to z is $\max_{w \in C_\varphi(z)} s(w)$. Note that C_φ can be computed in PTIME (see [10] for a proof).

Theorem 1: *Given (u, v), G and φ, it is in* PTIME *to compute the set C_φ of bidirectional scattered matches pivoted at (u, v).* □

Example 8: Consider φ_1 in Example 5. To illustrate, we focus on a subgraph pivoted at u and C_{φ_1} for $\rho = (x_0, x_1, x_2, x_3)$ in Figure 7.

The set C_{φ_1} is shown in Figure 7(b). Based on C_{φ_1}, one can compute the bound $\hat{s}(z)$ for each z in φ_1, by taking the maximum score of vertices in $C_{\varphi_1}(z)$, *e.g.,* $\hat{s}(x_1) = \max\{s(w_1), s(w_2)\} = 0.9$. □

5.3 Top-k Algorithm

We now present our top-k algorithm, denoted by TopkEx.

Overview. To get top-k evidences, we process REPs in Σ one by one. For each φ, we compute the score upper bound $\hat{s}(\varphi)$ (see Equation 1) for all possible evidences involving φ. Moreover, we maintain a heap Ψ of size k for top-k evidences found so far. Denote by T_k the k-th highest ranking score of evidences in Ψ. If $\hat{s}(\varphi) < T_k$, no witnesses of φ can contribute to the top-k, and thus, we stop processing φ. Otherwise, if $\hat{s}(\varphi) \geq T_k$, some witnesses of φ may make a top-k evidence. Then we *strategically* inspect the witnesses of φ, so that on the one hand, promising evidences are examined with high priority, and on the other hand, the score bounds can be gradually tightened and witnesses leading to low-ranked evidences are more likely to be pruned directly, without computing their exact scores.

If $\hat{s}(\varphi) \leq T_k$ for all REPs φ in Σ, we terminate the algorithm early and Ψ is returned as the desired set of top-k evidences.

Algorithm. Algorithm TopkEx implements this in Figure 8. It first sorts REPs in Σ in the decreasing order of rule importance (line 1). Then for each φ in Σ, we compute the set C_φ; as a by-product, this gives an upper bound $\hat{s}(z)$ for each variable z and an upper bound $\hat{s}(\varphi)$ for all possible witnesses h of φ (see Equation 1, lines 3-4). If $\hat{s}(\varphi) \geq T_k$ (*i.e.,* some witnesses of φ may make a top-k evidence), we select a "promising" vertex w in $C_\rho(z)$ via procedure SelectVertexUB (line 6) and process all witnesses h with $h(z) = w$ via procedure NextMatch (lines 7-9). The witnesses leading to highly ranked evidences will update Ψ and T_k (line 8). After all witnesses with $h(z) = w$ are processed, w is removed from $C_\varphi(z)$ and C_φ is refined accordingly, by checking the refinement conditions for each vertex in C_φ; this may in turn lead to a possibly tighter upper bound $\hat{s}(\varphi)$ (line 9). This process continues until $\hat{s}(\varphi) \leq T_k$ (line 5), *i.e.,* when none of not-yet-processed witnesses of φ can contribute to the top-k evidences. Then we stop the processing of φ. Finally, when the score upper bounds of *all* REPs in Σ are no larger than T_k, TopkEx terminates early, and Ψ is returned as the top-k set (line 10).

Procedure SelectVertexUB. Given a set C_φ, this procedure returns a vertex w in $C_\varphi(z)$. Recall that after processing all witnesses h with $h(z) = w$, this w will be eventually removed from $C_\varphi(z)$, and the bound $\hat{s}(z)$, which is defined to be the maximum score of vertices in $C_\varphi(z)$, can be possibly tightened. Therefore, we select w so that $\hat{s}(z)$ is reduced as much as possible and more witnesses are pruned early.

Specifically, for each $w \in C_\varphi(z)$ ($z \neq x_0, y_0$), we define an indicator Δ_w, expressing the reduction of $\hat{s}(z)$ if w is removed, *i.e.,*

$$\Delta_w = \max_{w' \in C_\varphi(z)} s(w') - \max_{w' \in C_\varphi(z) \setminus \{w\}} s(w'),$$

where the first (resp. second) term is the bound $\hat{s}(z)$ before (resp. after) w is removed from $C_\varphi(z)$. Then the vertex w with the maximum Δ_w is selected by SelectVertexUB as the next vertex to be processed.

Procedure NextMatch. Taking G, φ and the selected $w \in C_\varphi(z)$ as input, NextMatch returns witnesses h such that $h(z) = w$ one by one. For each path ρ where $z \in \rho$, its matching results can be obtained via depth first search (DFS) from w. We maintain the results for each path in designated structures and combine the results of all paths to generate the complete witnesses h of φ.

Example 9: Let the current Ψ have $k = 3$ and $T_k = 2.7$ (not shown). TopkEx first computes the bound $\hat{s}(z)$ for each z in φ_1 as in Example 8. Suppose after summing up $\hat{s}(z)$ for all z by Equation 1, the current score bound $\hat{s}(\varphi_1)$ is 2.9. Since $\hat{s}(\varphi_1) > T_k$, we call SelectVertexUB to get the vertex w_2 with the maximum Δ_w ($= \max_{w' \in \{w_1, w_2\}} s(w') - \max_{w' \in \{w_1\}} s(w') = s(w_2) - s(w_1) = 0.3$, Figure 7(c)). Three witnesses of ρ with $h(x_1) = w_2$ can be identified via DFS, *i.e.,* $h_1(\rho) = (u, w_2, w_5, w_8), h_2(\rho) = (u, w_2, w_5, w_9)$ and $h_3(\rho) = (u, w_2, w_6, w_9)$. Suppose we find that only $s(\varphi_1, h_1) = 2.9 > T_k$. We then update Ψ with (φ_1, h_1) and T_k gets tighter (2.9). Finally, we remove w_2 from $C_{\varphi_1}(x_1)$ and refine C_{φ_1}, *e.g.,* the removal of w_2 leads to the removal of w_6 from $C_{\varphi_1}(x_2)$ since w_2 is the only parent of w_6 that is in $C_{\varphi_1}(x_1)$ and thus, w_6 is no longer a bidirectional scattered match of x_2 (Figure 7(d)). The bound $\hat{s}(\varphi_1)$ is re-computed based on the updated C_{φ_1}. If $\hat{s}(\varphi_1)$ is now less than T_k, TopkEx stops processing φ_1 and proceeds to the next REP in Σ. When $\hat{s}(\varphi) \leq T_k$ for *all* REPs φ in Σ, TopkEx terminates early and returns Ψ. □

Complexity. Denote by c_{match} the unit cost for computing or refining the set C_φ of bidirectional scattered matches for a given φ; as remarked earlier, it is in PTIME by dynamic programming (Theorem 1). For each C_φ, it is refined each time when a vertex is removed (line 9 in Figure 8), leading to at most $|C_\varphi|_{\mathsf{max}}$ times of refinement on C_φ, where $|C_\varphi|_{\mathsf{max}}$ is the maximum size of C_φ for all φ which is also a polynomial as remarked earlier. Thus, for $|\Sigma|$ REPs in Σ, TopkEx takes $O(c_{\mathsf{match}}|\Sigma||C_\varphi|_{\mathsf{max}})$ time in total.

6 EXPERIMENTAL STUDY

Using real-life graphs, we experimentally evaluated the effectiveness and efficiency of our local and global explanations.

Experimental setting. We start with the experimental setting.

Datasets. We used four real-life graphs G: (1) MovieLens [33], a bi-directed graph for movie recommendation that has 21K vertices and 2.6M edges, (2) Yelp [7], a user-business review dataset with 119K vertices and 3.7M edges, (3) CiaoDVD [5], a user-movie rating dataset with 93K vertices and 4.6M edges, and (4) Lthing [86], a user-book review dataset with 589K vertices and 1.8M edges. Except the user-item interaction graphs, Yelp, CiaoDVD and Lthing also include social relationships among users. Besides, each graph was enriched by using relevant knowledge graphs, *e.g.,* Freebase.

Following the prior work [21, 36, 66], we randomly picked 70%/10%/20% interaction history of each user in each dataset as the training/validation/testing set. For each observed user-item interaction pair, we treated it as a positive instance and did negative sampling to pair the user with an item that s/he did not interact before.

GNN models. We selected three GNN-based recommendation models $\mathcal{M}$ on heterogeneous graphs: (1) PinSAGE [81], a widely-used model that generates vertex embeddings by sampling and aggregating features from a vertex's neighborhoods. (2) HGT [36], a transformer-based method that designs a vertex- and edge-type dependent attention mechanism to handle the graph heterogeneity. (3) KGAT [66], a classical model that applies an attentive neighborhood aggregation mechanism on holistic graphs to learn user/item representations. Among these, PinSAGE is a basic GNN model while HGT and KGAT are representative GNN models incorporating KGs.

Rules. For PinSAGE, HGT and KGAT, we discovered 98, 99 and 145 REPs on MovieLens, 52, 46 and 86 REPs on Yelp, 172, 141 and 158 REPs on CiaoDVD, and 79, 56 and 79 REPs on Lthing, respectively.

Baselines. We implemented Makex in Python and C++, adopting Pytorch for the deep learning-related computations.

We compared with four baselines for local explanations: (1) GNNExplainer [82], (2) PGExplainer [49], (3) SubgraphX [85] and (4) PGMExplainer [63]. GNNExplainer extracts important subgraphs and vertex features by maximizing mutual information between subgraphs and predictions; PGExplainer outputs subgraphs as factual explanations by adopting a deep neural network to parameterize generation of explanations; SubgraphX returns subgraphs based on Shapley values; and PGMExplainer returns subgraphs of conditional probabilities by exploiting a Bayesian network.

We tested two baselines for global explanations: DAG [50] and $\mathsf{DAG}_{\mathsf{mini}}$, both of which output a set $\mathcal{S}_{\mathsf{sg}}$ of subgraphs as global explanations. DAG uses a randomized greedy algorithm to find global explanations that approximately fit an objective function. Since DAG runs out of memory on all datasets, we adapted it to $\mathsf{DAG}_{\mathsf{mini}}$ for generating candidate graph patterns with sampling strategies.

We adapted these to link prediction on heterogeneous graphs. We did not pick other methods (see the related work) as baselines since they either publish no source code or are infeasible to be adapted for link prediction tasks, *e.g.,* GraphMask [59] is not compared since it focuses on star graphs and question answering tasks, rather than link prediction; XGNN [83] is not compared since it takes as the input the manually-designed graph patterns, which requires the pre-knowledge about specific datasets; and GNNInterpreter [69] is tailored for graph classification tasks and it requires to feed into memory all graphs from the target class in the training set; it renders out of memory when it is adapted for link prediction since it needs to construct the K-hop neighborhood for millions of user-item pairs.

Default parameters. By default, we set the support threshold σ (resp. confidence δ) in RepsLearner as 150K, 50K, 1.5K and 50K (resp. 0.6, 0.6, 0.55 and 0.6) on MovieLens, Yelp, CiaoDVD and Lthing, respectively. We mined REPs in which patterns have at most 10 vertices and 8 edges. The value of N for precondition mining is 3. For each model on each dataset, we set (a) $k = 1$ in top-k local explanations, (b) $\alpha = 0.8$ for random walks, and (c) the weights $w_s = 0.3$ and $w_d = 0.7$. Besides, each model is best-tuned on each dataset, and the threshold for CTR predictions are selected by using validation data.

Environment. The experiments were conducted on a single machine powered by 256GB RAM, 32 processors with Intel(R) Xeon(R) Gold 5320 CPU @2.20GHz and one NVIDIA GeForce RTX 3090 GPU with 25 GB memory. For the precondition mining in RepsLearner, we used 25 threads for parallelism. For the lack of space, we report results on some datasets; the results on the others are consistent.

Evaluation metrics. To evaluate the effectiveness of local explanations [82, 84, 85], we used *Fidelity*, *Sparsity* and *Feature ratio*. Each metric was evaluated on M randomly selected testing pairs (u, v) for which $\mathcal{M}$ recommends item v to user u, where we set $M = 1,000$.

(1) Fidelity. It measures whether a local explanation is faithful to the model's predictions, by inspecting the structures and features identified by an explanation. For each testing pair (u, v), it checks whether the GNN prediction on the explanation minimally differs

Table 2: Local effectiveness (Top-1)

GNN Models	Methods	MovieLens		Yelp		CiaoDVD		Lthing	
		fidelity	sparsity	fidelity	sparsity	fidelity	sparsity	fidelity	sparsity
PinSAGE	PGExplainer	0.469	0.0426%	0.487	0.00494%	0.492	0.00351%	0.384	0.0284%
	GNNExplainer	0.539	2.99%	0.782	0.0474%	0.461	0.0712%	0.559	0.00756%
	SubgraphX	0.785	0.352%	0.818	0.352%	0.480	0.309%	0.671	0.032%
	PGMExplainer	0.724	0.265%	0.763	0.044%	0.873	0.644%	0.692	0.0245%
	Makex (**ours**)	**0.914**	**0.000561%**	**1.0**	**0.000303%**	**0.921**	**0.000496%**	**0.774**	**0.000163%**
HGT	PGExplainer	0.598	0.0330%	0.628	0.00425%	0.781	0.00524%	0.102	0.015%
	GNNExplainer	0.638	2.70%	0.702	0.0486%	0.499	0.0592%	0.424	0.00212%
	SubgraphX	0.791	0.417%	0.539	0.370%	0.466	0.205%	0.540	0.0197%
	PGMExplainer	0.743	0.0504%	0.734	0.0274%	0.550	0.153%	0.476	0.00504%
	Makex (**ours**)	**0.985**	**0.000279%**	**0.997**	**0.000154%**	**0.887**	**0.000235%**	**1.0**	**0.000166%**
KGAT	PGExplainer	0.480	0.0259%	0.607	0.00427%	0.553	0.00802%	0.110	0.019%
	GNNExplainer	0.566	2.90%	0.645	0.0130%	0.382	0.196%	0.685	0.00211%
	SubgraphX	0.814	0.359%	0.762	0.0918%	0.446	0.380%	0.496	0.0241%
	PGMExplainer	0.800	0.0314%	0.729	0.0216%	0.590	0.143%	0.687	0.00467%
	Makex (**ours**)	**0.862**	**0.000254%**	**0.823**	**0.000154%**	**0.759**	**0.000243%**	**0.790**	**0.000689%**

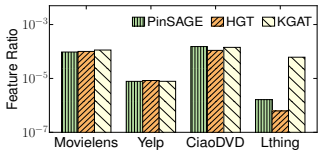


Figure 9: Feature ratio (Top-1)

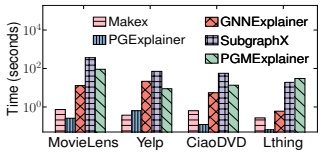


Figure 10: Local efficiency (Top-1)

from that on the original graph. We define its fidelity as follows:

$$\text{fidelity@}k = \frac{1}{M} \cdot \frac{1}{k} \sum_{(u,v)} \sum_{i=1}^{k} \mathbb{1}(\hat{y}^{(i)}_{(u,v)} = y_{(u,v)}),$$

where $\hat{y}^{(i)}_{(u,v)}$ (resp. $y_{(u,v)}$) is the prediction for (u, v) on the subgraph induced from the i-th explanation (resp. the original graph), and $\mathbb{1}(\cdot)$ is an indicator function that returns 1 if $\hat{y}^{(i)}_{(u,v)} = y_{(u,v)}$.

(2) Sparsity. It measures the ratio of edges selected by a local explanation to all the edges in the entire graph. We define

$$\text{sparsity@}k = \frac{1}{M} \cdot \frac{1}{k} \sum_{(u,v)} \sum_{i=1}^{k} \frac{\#\text{selected\_edges}^{(i)}_{(u,v)}}{|E|},$$

where $\#\text{selected\_edges}^{(i)}_{(u,v)}$ is the number of edges in the i-th explanation for (u, v) and $|E|$ is the number of edges in original graph G.

Intuitively, higher fidelity indicates that more discriminative structures/features are identified; lower sparsity indicates that explanations tend to capture the most important information only.

(3) Feature ratio. We also report the feature ratio, *i.e.,* the average ratio of features used in an explanation to the total number of features.

For global explanations, we used another two popular metrics following [50, 83]: (1) *overall recognizability* that checks the recognizability of GNN models on the set $\mathcal{E}$ of global explanations (*i.e.,* $\mathcal{E} = \Sigma$ and $\mathcal{E} = \mathcal{S}_{\text{sg}}$ for Makex and DAG_{mini}, respectively), *i.e.,*

$$\text{recognizability} = \frac{\sum_{\text{exp} \in \mathcal{E}} \hat{y}(\text{exp})}{|\mathcal{E}|},$$

where exp is an explanation in $\mathcal{E}$ (*e.g.,* a canonical graph generated from REP φ in Σ; see [10] for details), and $\hat{y}(\text{exp})$ is the predicted label of the GNN by feeding exp into the model; and (2) *reliability* that computes the coverage of $\mathcal{E}$ on all testing pairs, *i.e.,*

$$\text{reliability} = \frac{|\bigcup_{\text{exp} \in \mathcal{E}} \text{testPairs(exp)}|}{\#\text{totalPairs}}.$$

where #totalPairs and testPairs (exp) are the total number of testing pairs and the set of testing pairs that can be explained by exp (*e.g.,* (u, v) can be explained by φ if φ is applicable at (u, v)), respectively.

Experimental results. We next report our findings.

Exp-1: Effectiveness of local explanations. We first tested the effectiveness of local explanations and present the sensitivity test.

Effectiveness of top-1 explanation. As shown in Table 2, Makex is more accurate than the baselines in fidelity over the four datasets. On average, the fidelity of Makex is 0.893, 181.42%, 61.64%, 47.86% and 31.56% higher than PGExplainer, GNNExplainer, SubgraphX and PGMExplainer, respectively, up to 880.39%, 135.85%, 91.88% and 110.08%. This is because Makex finds not only important subgraphs but also discriminative features that SubgraphX, PGExplainer and PGMExplainer fail to find, and GNNExplainer masks the same features for all vertices (*i.e.,* the features on different vertices are of equal importance in its explanations, which is, however, often not true). The sparsity of Makex outperforms all baselines by 3 orders of magnitude on average, up to 4 orders of magnitude.

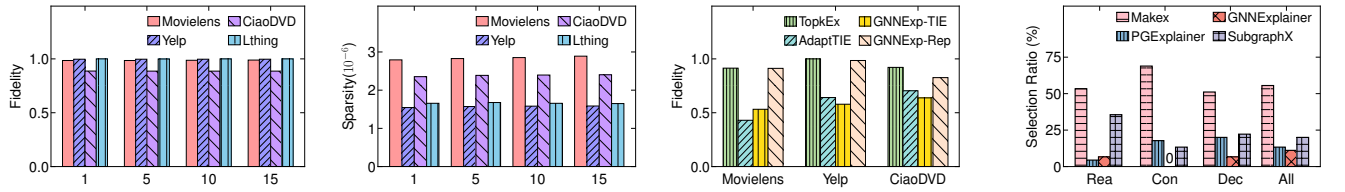
For feature ratios shown in Figure 9, Makex uses a small number of features to explain GNNs on all datasets, and highlights important features for each vertex. In contrast, SubgraphX, PGExplainer and PGMExplainer do not consider features; GNNExplainer selects the same subset of features for all vertices (not shown).

We also tested the sensitivity of Makex to various parameters.

Varying k. We varied k in Makex from 1 to 15 for HGT model, to test its impact on local explanations. As shown in Figure 11(a), the fidelity of HGT model on each dataset remains steady when k increases, indicating the effectiveness of ranking scores, *i.e.,* more important explanations get higher priority. As shown in Figure 11(b), when k increases, the sparsity of HGT also remains steady since TopkEx prioritizes succinct patterns. For feature ratios (not shown), k has little impact since TopkEx prioritizes explanations by the decisiveness of their features, instead of the number of features used.

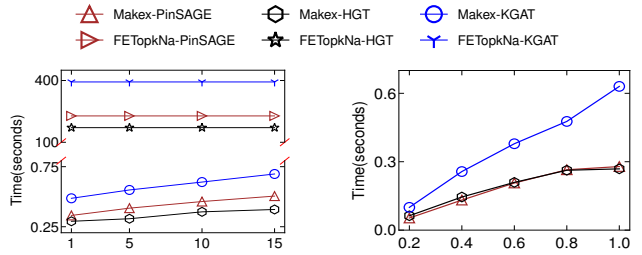
Impact of REPs. To show the advantages of REPs over existing graph rules, *e.g.,* TIEs [21], for local explanations, we also compared our TopkEx algorithm with (1) an adapted version of TopkEx, by replacing REPs with TIEs, termed AdaptTIE, and (2) two adapted versions of GNNExplainer, by replacing its input with smaller subgraphs deduced by TIEs and REPs, respectively, termed GNNExp-TIE and GNNExp-Rep. As shown in Figure 11(c), for PinSAGE model, the fidelity of TopkEx outperforms all competitors on all datasets. since REPs treat a GNN model $\mathcal{M}$ as their consequence and support the 1-WL test, which TIEs fail to do, and GNNExplainer cannot discriminate distinct feature effect on individual vertex. Besides, the fidelity of TopkEx is higher than GNNExp-Rep by up to 11.50%, verifying the effectiveness of the ranking strategy in TopkEx.

User study. We conducted a user study on Amazon Mechanical Turk (Mturk) [9], by inviting participants to indicate which explanation in Figure 4 is better. We required each participant to be a *master worker* (*i.e.,* those have demonstrated excellency across a wide range of tasks) in Mturk, and did not assume that they have background on ML or GNN recommendation. Therefore, we interpreted each explanation in a plain and simple language to ensure that the explanation is understandable for average persons (see the online survey in [11]). Another user study about restaurant recommendation is reported in [10]. We adopted the following four metrics: (a) *Reasonableness* (Rea): Is the logic behind the explanation reasonable? (b) *Conciseness* (Con): Is the information contained in the



(a) Fidelity of HGT: varying k (b) Sparsity of HGT: varying k (c) Impact of REPs for PinSAGE (d) User Study

Figure 11: Local sensitivity (Effectiveness)



(a) Yelp: varying k (Time) (b) CiaoDVD: varying $|G|$

Figure 12: Local sensitivity (Efficiency)

explanation concise? (c) *Decisiveness* (Dec): Does the explanation only contain decisive factors for explaining the recommendation? (d) *Overall* (All) that considers all above metrics simultaneously.

To avoid random choices by participants, we cross-checked the validity of answers of each participant. More specifically, we presented two questions for each metric, which asked the participants to select the best and best two explanations *w.r.t.* each metric, respectively. If the top-2 explanations do not include the best one, the response was marked as invalid. We received 58 responses in total, out of which 45 are valid after cross-checking. The results are shown in Figure 11(d), where 55.56% responses indicate that the explanation of Makex has the best overall performance. GNNExplainer and PGExplainer got low scores in Rea, due to the fragmentation (*e.g.,* u and v are disconnected) in their explanations. In contrast, many users indicated that SubgraphX is reasonable, but its explanation is the largest (*i.e.,* the lowest Con), which is too dense to digest. Note that although both SubgraphX and Makex use MCTS for discovering important subgraphs, the explanation of Makex is not necessarily a subgraph of SubgraphX (*e.g.,* path (u, m_1, t_1) in Makex is not in SubgraphX, Figure 4), since SubgraphX returns a subgraph for a *specific* user-item pair while Makex applies a (top-ranked) REP that is composed of selected paths and preconditions faithful to the given GNN model; each REP is constructed by using *multiple* user-item pairs with MCTS. Besides, Makex utilizes 1WL predicates to avoid the isomorphic neighborhood information.

Exp-2: Efficiency of local explanations. We next evaluated the efficiency of TopkEx, by comparing it with an additional variant $\text{FETopk}_{\text{na}}$, which finds all witnesses for each REP in Σ, sorts the evidences by scores, and returns the top-k ones. Using the default REPs Σ and $k = 1$, we report the average time for each testing pair.

Efficiency. As shown in Figure 10, Makex is 75.8X faster than all the baselines on average. On Yelp, its top-1 explanation takes 0.38s on average. Although PGExplainer takes less time than Makex on MovieLens, CiaoDVD, and Lthing datasets since it pre-trains a neural network offline, it has worse fidelity as shown in Table 2.

Varying k. As shown in Figures 12(a) by varying k from 1 to 15 on Yelp, $\text{FETopk}_{\text{na}}$ is indifferent to k, since it enumerates all evidences from G regardless of k. In contrast, TopkEx takes longer when k gets larger since more evidences are checked, as expected. On Yelp, TopkEx is 575X faster than $\text{FETopk}_{\text{na}}$, up to 808X, verifying the effectiveness of its pruning strategies for top-k explanations.

Varying $|G|$. We varied the scaling factor of G from 20% to 100% in Figure 12(b). All methods take longer given a larger graph, For PinSAGE, TopkEx is 4.2X slower when $|G|$ is from 20% to 100%.

Exp-3: Effectiveness of global explanations.

Effectiveness. In Figure 13(a), the average recognizability of Makex on the three models is 72% and 24% (percentage points) higher than DAG_{mini} on MovieLens and CiaoDVD, respectively, and DAG runs out of memory. Similarly, the reliability of Makex is 95% and 30% better than DAG_{mini} on MovieLens and CiaoDVD in Figure 13(b), respectively. It is because REPs discovery is guided by the given GNN with 1-WL test, while DAG_{mini} depends heavily on the candidate explanations mined by the frequent pattern mining method.

Varying #trainingPairs. We varied the ratio of training pairs used for pattern generation from 25% to 100% in Figure 14(a). When few training pairs are retained, some recommendation scenarios are not covered, leading to low reliability. However, by using all training pairs, the reliability of Makex is high, *e.g.,* 1.0 for HGT on Yelp, indicating that Makex can cover different cases in practice.

Varying N. We varied the number N of candidate preconditions for each path from 1 to 5 in Figure 14(b). With larger N, the reliability of Σ gets better, since there are more REPs that satisfy the support/confidence thresholds, covering more testing pairs. We find the reliability of Σ becomes steady for $N \geq 3$. In contrast, since the recognizability is mainly impacted by the expressivity of REPs, not the number of REPs in Σ, it is fluctuant as N gets larger (not shown).

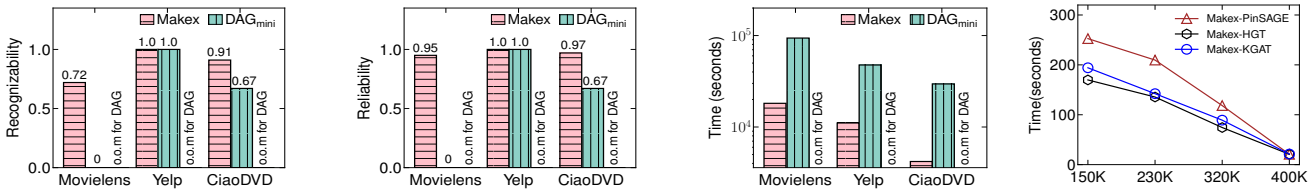
Exp-4: Efficiency of global explanations. We next evaluated the efficiency of RepsLearner for providing global explanations.

Efficiency. In Figure 13(c), RepsLearner is up to 7X faster than the baselines on all datasets for HGT. On CiaoDVD, it takes 1.17 hours to find all REPs while DAG_{mini} takes 8 hours. Since DAG_{mini} fails to finish in 3 days on Yelp, we sampled 10% of training pairs for DAG_{mini}, which, however, still takes longer than RepsLearner.

Varying σ and δ. Varying support threshold σ from 150K to 400K on MovieLens, we report the runtime of the precondition mining phase of RepsLearner in Figure 13(d). When σ increases, it runs faster since larger σ can filter more candidate rules by the anti-monotonicity of support. In contrast, given a larger confidence threshold δ, RepsLearner gets slightly slower (not shown) since confidence is not anti-monotonic, and hence more rules are checked.

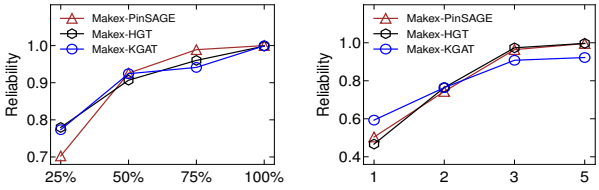
Varying N, α_1 and α_2. RepsLearner takes longer when N, α_1 or α_2 gets larger, as expected, since more REPs are generated (not shown).

Summary. We find the following. (1) Makex gives effective local explanations. Over four real-life graphs, its average fidelity and



(a) Recognizability (Global) (b) Reliability (Global) (c) Time cost of HGT (Global) (d) MovieLens: varying σ (Time)

Figure 13: Global effectiveness and efficiency



(a) Yelp: varying #training (b) MovieLens: varying N

Figure 14: Global sensitivity (Reliability)

sparsity of top-1 explanations are 0.893 and 0.00225%, 80.62% and 3 orders of magnitude better than the prior methods on average, respectively, with a small feature ratio. (2) Makex is efficient. For each user-item pair, it takes 0.38s on average to generate top-1 local explanations on a graph with 119K vertices and 3.7M edges. (3) TopkEx is 575X faster than FETopk$_{na}$ on average on Yelp, up to 808X, verifying the effectiveness of our top-k method. (4) The set Σ of REPs provides good global explanations. Its recognizability and reliability for global explanation are 32% and 42% higher than the baselines on average, up to 72% and 95%, respectively. (5) RepsLearner is faster than the prior global explanation methods by up to 7X.

7 RELATED WORK

Explanation methods. Prior methods are classified as follows.

(1) Ante-hoc self-explainable models. Such models provide explanations by exploiting knowledge graphs (KGs), *e.g.,* Ripplenet [64], KPRN [70] and TMER [18] identify human-designed meta-paths between users and items; RuleRec [53], PGPR [77] and KGIN [68] discover meta-paths from KGs. Review information has also been used, *e.g.,* KEGNN [51] adopts the Gate Recurrent Unit (GRU) [20] to generate textual explanation from KGs, and TGNN [60] devises a sentence-enhanced topic graph from reviews. However, such a GNN model cannot be extended to other GNN models once trained.

(2) Post-hoc explanation methods. These methods can be classified as follows. (a) Utilizing internal parameters, which are not always accessible; such methods, *e.g.,* CAM [55] and Sensitivity Analysis (SA) [12], are not suitable for explaining recommendations. (b) Utilizing surrogate models; *e.g.,* GraphLime [38] selects vertices and their K-hop neighborhood by utilizing Hilbert-Schmidt Independence Criterion Lasso [80], and PGMExplainer [63] exploits a Bayesian network. (c) Utilizing local information, which finds *important subgraphs* as explanations by first perturbing the input graph and then iteratively selecting the subgraph with the maximum information gain, *e.g.,* GNNExplainer [82], GraphMask [59], PGExplainer [49], SubgraphX [85] and Zorro [28]. However, these methods cannot tell precisely what features are decisive for an ML model to make predictions and under what conditions the predictions can be made. Moreover, they provide only local explanations.

(3) Rule-based methods. [17, 58] explain negative predictions with rules of conjunction of constant predicates, which are globally-consistent for all training data. However, they target relational data and do not work on GNNs since they omit graph topology.

(4) Global explanations. Global explanations include, *e.g.,* GNNInterpreter [69] via a numerical optimization approach, XGNN [83] using reinforcement learning and DAG [50] with a randomized greedy algorithm. However, these methods either require pre-processed graph patterns as input or assume sub-structures for explanation as a Gilbert random graph, which are often beyond reach in practice.

Makex differs from prior work in the following. (a) It proposes a logic approach to explaining why a GNN-based model $\mathcal{M}$ makes a recommendation, by disclosing both graph patterns and logic predicates to reveal the correlations, interactions and dependencies of features, rather than meta paths or subgraphs (with one-size-fit-all features). (b) It develops a method for automatically learning a combination of patterns and predicates to explain $\mathcal{M}(x, y)$. (c) It provides both local and global explanations for $\mathcal{M}(x, y)$. (d) It offers higher fidelity and lower sparsity, as shown in Section 6. (e) It aims to explain different behaviors of $\mathcal{M}$ via 1-WL predicate.

Rules for graphs. Association rules have been studied to catch the regularity among entities in graphs, *e.g.,* GPARs [26], GFDs [27] and GEDs [25], GARs [23] and TACOs [24]. To reduce the cost, GCRs [22] and TIEs [21] adopt star patterns and restricted predicates, where GCRs focus on graph cleaning and TIEs aim to reduce false positives and false negatives of ML recommendations.

As opposed to the prior work, (1) for a given ML model $\mathcal{M}$, REPs treat $\mathcal{M}$ as the consequence of the rules, in order to discover explanations for the predictions of $\mathcal{M}$. (2) REPs embed the 1-WL test [72] as a predicate, to offer sufficient expressive power for GNN recommendation models that are based on the 1-WL test [35].

8 CONCLUSION

The novelty of Makex includes (1) REPs to (a) provide both global and local explanations for GNN recommendations, (b) reveal not only decisive topology and features but also conditions under which the ML predictions are made, and (c) explain different behaviors of GNN predictions via, *e.g.,* 1-WL test; (2) a GNN-guided algorithm for discovering REPs pertaining to a given model $\mathcal{M}$; and (3) an algorithm for generating top-ranked local explanations. Our experimental study has verified that Makex is promising in practice.

One topic for future work is to develop explanations for negative ML prediction, *i.e.,* bad outcomes. Another topic is to produce explanation for fairness debugging [56], to explain biased behaviors.

ACKNOWLEDGMENTS

This work was supported by the National Natural Science Foundation of China (62372030, 62302027, 62202313), Guangdong Basic and Applied Basic Research Foundation 2022A1515010120, Longhua Science & Technology Innovation Bureau 11501A20240704A24CA6C.

REFERENCES

[1] 1995. A Walk in the Clouds. https://www.imdb.com/title/tt0114887/?ref_=fn_al_tt_1.

[2] 1997. Titanic. https://www.imdb.com/title/tt0120338/?ref_=nv_sr_srsg_0_tt_5_nm_3_q_tit.

[3] 2017. Amazon GNN Recommender. *https://www.aboutamazon.in/news/workplace/how-amazons-augmented-graph-neural-networks-helps-sellers-get-insights-into-consumer-preferences//.*

[4] 2017. Amazon GNN Recommender in Product. *https://www.amazon.science/blog/using-graph-neural-networks-to-recommend-related-products//.*

[5] 2017. CiaoDVD movie ratings. *http://konect.cc/networks/librec-ciaodvd-movie_ratings/.*

[6] 2017. Uber GNN Recommender. *https://www.uber.com/en-HK/blog/uber-eats-graph-learning//.*

[7] 2021. Yelp dataset. *https://www.yelp.com/dataset/.*

[8] 2023. Save on Mac or iPad for college. https://www.apple.com/us-edu/store.

[9] 2024. Amazon Mechanical Turk. *https://www.mturk.com/.*

[10] 2024. Code, datasets and full version. https://github.com/SICS-Fundamental-Research-Center/Makex.

[11] 2024. Survey on Best Explanations for Recommendation. https://forms.gle/c8P1sJQGkKUxiwjm9.

[12] Federico Baldassarre and Hossein Azizpour. 2019. Explainability Techniques for Graph Convolutional Networks. In *ICML Workshop "Learning and Reasoning with Graph-Structured Representations"*.

[13] Avishek Bose and William Hamilton. 2019. Compositional fairness constraints for graph embeddings. In *ICML*. PMLR, 715–724.

[14] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. 2012. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games* 4, 1 (2012), 1–43.

[15] Jianxin Chang, Chen Gao, Xiangnan He, Depeng Jin, and Yong Li. 2020. Bundle recommendation with graph convolutional networks. In *SIGIR*. 1673–1676.

[16] Jianxin Chang, Chen Gao, Yu Zheng, Yiqun Hui, Yanan Niu, Yang Song, Depeng Jin, and Yong Li. 2021. Sequential recommendation with graph neural networks. In *SIGIR*. 378–387.

[17] Chaofan Chen, Kangcheng Lin, Cynthia Rudin, Yaron Shaposhnik, Sijia Wang, and Tong Wang. 2018. An Interpretable Model with Globally Consistent Explanations for Credit Risk. *CoRR* abs/1811.12615 (2018). arXiv:1811.12615 http://arxiv.org/abs/1811.12615

[18] Hongxu Chen, Yicong Li, Xiangguo Sun, Guandong Xu, and Hongzhi Yin. 2021. Temporal Meta-path Guided Explainable Recommendation. In *WSDM*. ACM, 1056–1064.

[19] Tianwen Chen and Raymond Chi-Wing Wong. 2020. Handling information loss of graph neural networks for session-based recommendation. In *SIGKDD*. 1172–1180.

[20] Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. In *NIPS 2014 Workshop on Deep Learning, December 2014*.

[21] Lihang Fan, Wenfei Fan, Ping Lu, Chao Tian, and Qiang Yin. 2024. Enriching Recommendation Models with Logic Conditions. *Proc. ACM Manag. Data* (2024).

[22] Wenfei Fan, Wenzhi Fu, Ruochun Jin, Muyang Liu, Ping Lu, and Chao Tian. 2023. Making It Tractable to Catch Duplicates and Conflicts in Graphs. *Proc. ACM Manag. Data* 1, 1 (2023), 86:1–86:28.

[23] Wenfei Fan, Ruochun Jin, Muyang Liu, Ping Lu, Chao Tian, and Jingren Zhou. 2020. Capturing Associations in Graphs. *PVLDB* 13, 11 (2020), 1863–1876.

[24] Wenfei Fan, Ruochun Jin, Ping Lu, Chao Tian, and Ruiqi Xu. 2022. Towards Event Prediction in Temporal Graphs. *PVLDB* 15, 9 (2022), 1861–1874.

[25] Wenfei Fan and Ping Lu. 2019. Dependencies for Graphs. *ACM Trans. Database Syst.* 44, 2 (2019), 5:1–5:40.

[26] Wenfei Fan, Xin Wang, Yinghui Wu, and Jingbo Xu. 2015. Association Rules with Graph Patterns. *PVLDB* 8, 12 (2015), 1502–1513.

[27] Wenfei Fan, Yinghui Wu, and Jingbo Xu. 2016. Functional dependencies for graphs. In *SIGMOD*. ACM, 1843–1857.

[28] Thorben Funke, Megha Khosla, Mandeep Rathee, and Avishek Anand. 2022. Z orro: Valid, sparse, and stable explanations in graph neural networks. *IEEE Transactions on Knowledge and Data Engineering* (2022).

[29] Chen Gao, Yu Zheng, Nian Li, Yinfeng Li, Yingrong Qin, Jinghua Piao, Yuhan Quan, Jianxin Chang, Depeng Jin, Xiangnan He, and Yong Li. 2023. A survey of graph neural networks for recommender systems: Challenges, methods, and directions. *ACM Transactions on Recommender Systems* 1, 1 (2023), 1–51.

[30] Michael Garey and David Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company.

[31] Martin Grohe. 2020. word2vec, node2vec, graph2vec, X2vec: Towards a Theory of Vector Embeddings of Structured Data. In *PODS*. ACM, 1–16.

[32] Martin Grohe. 2021. The Logic of Graph Neural Networks. In *LICS*. 1–17.

[33] F. Maxwell Harper and Joseph A. Konstan. 2016. The MovieLens Datasets: History and Context. *ACM Trans. Interact. Intell. Syst.* 5, 4 (2016), 19:1–19:19.

[34] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yongdong Zhang, and Meng Wang. 2020. LightGCN: Simplifying and powering graph convolution network for recommendation. In *SIGIR*. 639–648.

[35] Yang Hu, Xiyuan Wang, Zhouchen Lin, Pan Li, and Muhan Zhang. 2022. Two-Dimensional Weisfeiler-Lehman Graph Neural Networks for Link Prediction. *CoRR* abs/2206.09567 (2022).

[36] Ziniu Hu, Yuxiao Dong, Kuansan Wang, and Yizhou Sun. 2020. Heterogeneous graph transformer. In *The Web conference 2020*. 2704–2710.

[37] Chao Huang, Huance Xu, Yong Xu, Peng Dai, Lianghao Xia, Mengyin Lu, Liefeng Bo, Hao Xing, Xiaoping Lai, and Yanfang Ye. 2021. Knowledge-aware coupled graph neural network for social recommendation. In *AAAI*, Vol. 35. 4115–4122.

[38] Qiang Huang, Makoto Yamada, Yuan Tian, Dinesh Singh, and Yi Chang. 2022. GraphLIME: Local Interpretable Model Explanations for Graph Neural Networks. *TKDE* 35, 7 (2022), 6968–6972.

[39] Wensen Jiang, Yizhu Jiao, Qingqin Wang, Chuanming Liang, Lijie Guo, Yao Zhang, Zhijun Sun, Yun Xiong, and Yangyong Zhu. 2022. Triangle graph interest network for click-through rate prediction. In *WSDM*. 401–409.

[40] Bowen Jin, Chen Gao, Xiangnan He, Depeng Jin, and Yong Li. 2020. Multi-behavior recommendation with graph convolutional networks. In *SIGIR*. 659–668.

[41] Jiarui Jin, Jiarui Qin, Yuchen Fang, Kounianhua Du, Weinan Zhang, Yong Yu, Zheng Zhang, and Alexander J Smola. 2020. An efficient neighborhood-based interaction model for recommendation on heterogeneous graph. In *SIGKDD*. ACM, 75–84.

[42] Pigi Kouki, James Schaffer, Jay Pujara, John O'Donovan, and Lise Getoor. 2019. Personalized explanations for hybrid recommender systems. In *International Conference on Intelligent User Interfaces*. 379–390.

[43] Johannes Kunkel, Tim Donkers, Lisa Michael, Catalin-Mihai Barbu, and Jürgen Ziegler. 2019. Let me explain: Impact of personal and impersonal explanations on trust in recommender systems. In *CHI conference on human factors in computing systems*. 1–12.

[44] Zekun Li, Zeyu Cui, Shu Wu, Xiaoyu Zhang, and Liang Wang. 2019. Fi-GNN: Modeling feature interactions via graph neural networks for CTR prediction. In *CIKM*. 539–548.

[45] Dandan Lin, Shijie Sun, Jingtao Ding, Xuehan Ke, Hao Gu, Xing Huang, Chonggang Song, Xuri Zhang, Lingling Yi, Jie Wen, and Chuan Chen. 2022. PlatoGL: Effective and scalable deep graph learning system for graph-enhanced real-time recommendation. In *CIKM*. 3302–3311.

[46] Meng Liu, Jianjun Li, Guohui Li, and Peng Pan. 2020. Cross domain recommendation via bi-directional transfer graph collaborative filtering networks. In *CIKM*. 885–894.

[47] Weiwen Liu, Qing Liu, Ruiming Tang, Junyang Chen, Xiuqiang He, and Pheng Ann Heng. 2020. Personalized Re-ranking with Item Relationships for E-commerce. In *CIKM*. 925–934.

[48] Xin Liu, Zheng Li, Yifan Gao, Jingfeng Yang, Tianyu Cao, Zhengyang Wang, Bing Yin, and Yangqiu Song. 2024. Enhancing User Intent Capture in Session-Based Recommendation with Attribute Patterns. *Advances in Neural Information Processing Systems* 36 (2024).

[49] Dongsheng Luo, Wei Cheng, Dongkuan Xu, Wenchao Yu, Bo Zong, Haifeng Chen, and Xiang Zhang. 2020. Parameterized explainer for graph neural network. *NeurIPS* 33 (2020), 19620–19631.

[50] Ge Lv and Lei Chen. 2023. On Data-Aware Global Explainability of Graph Neural Networks. *PVLDB* 16, 11 (2023), 3447–3460.

[51] Ziyu Lyu, Yue Wu, Junjie Lai, Min Yang, Chengming Li, and Wei Zhou. 2022. Knowledge enhanced graph neural networks for explainable recommendation. *IEEE Transactions on Knowledge and Data Engineering* 35, 5 (2022), 4954–4968.

[52] Chen Ma, Liheng Ma, Yingxue Zhang, Jianing Sun, Xue Liu, and Mark Coates. 2020. Memory augmented graph neural networks for sequential recommendation. In *AAAI*, Vol. 34. 5045–5052.

[53] Weizhi Ma, Min Zhang, Yue Cao, Woojeong Jin, Chenyang Wang, Yiqun Liu, Shaoping Ma, and Xiang Ren. 2019. Jointly Learning Explainable Rules for Recommendation with Knowledge Graph. *CoRR* abs/1903.03714 (2019). http://arxiv.org/abs/1903.03714

[54] Christopher Morris, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. 2019. Weisfeiler and Leman Go Neural: Higher-Order Graph Neural Networks. In *AAAI*. 4602–4609.

[55] Phillip E Pope, Soheil Kolouri, Mohammad Rostami, Charles E Martin, and Heiko Hoffmann. 2019. Explainability methods for graph convolutional neural networks. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 10772–10781.

[56] Romila Pradhan, Jiongli Zhu, Boris Glavic, and Babak Salimi. 2022. Interpretable data-based explanations for fairness debugging. In *SIGMOD*. 247–261.

[57] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. "Why should I trust you?" Explaining the predictions of any classifier. In *SIGKDD*. 1135–1144.

[58] Cynthia Rudin and Yaron Shaposhnik. 2023. Globally-Consistent Rule-Based Summary-Explanations for Machine Learning Models: Application to Credit-Risk

Evaluation. *J. Mach. Learn. Res.* 24 (2023), 16:1–16:44.
[59] Michael Sejr Schlichtkrull, Nicola De Cao, and Ivan Titov. 2020. Interpreting Graph Neural Networks for NLP With Differentiable Edge Masking. In *ICLR*.
[60] Jie Shuai, Le Wu, Kun Zhang, Peijie Sun, Richang Hong, and Meng Wang. 2023. Topic-enhanced Graph Neural Networks for Extraction-based Explainable Recommendation. In *SIGIR*. 1188–1197.
[61] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy P. Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. 2017. Mastering the game of go without human knowledge. *Nature* 550, 7676 (2017), 354–359.
[62] Jianing Sun, Wei Guo, Dengcheng Zhang, Yingxue Zhang, Florence Regol, Yaochen Hu, Huifeng Guo, Ruiming Tang, Han Yuan, Xiuqiang He, and Mark Coates. 2020. A Framework for Recommending Accurate and Diverse Items Using Bayesian Graph Convolutional Neural Networks. In *KDD*. ACM, 2030–2039.
[63] Minh Vu and My T Thai. 2020. Pgm-explainer: Probabilistic graphical model explanations for graph neural networks. *Advances in neural information processing systems* 33 (2020), 12225–12235.
[64] Hongwei Wang, Fuzheng Zhang, Jialin Wang, Miao Zhao, Wenjie Li, Xing Xie, and Minyi Guo. 2018. RippleNet: Propagating User Preferences on the Knowledge Graph for Recommender Systems. In *CIKM*. ACM, 417–426.
[65] Jizhe Wang, Pipei Huang, Huan Zhao, Zhibo Zhang, Binqiang Zhao, and Dik Lun Lee. 2018. Billion-scale commodity embedding for e-commerce recommendation in alibaba. In *SIGKDD*. ACM, 839–848.
[66] Xiang Wang, Xiangnan He, Yixin Cao, Meng Liu, and Tat-Seng Chua. 2019. KGAT: Knowledge Graph Attention Network for Recommendation. In *SIGKDD*. ACM, 950–958.
[67] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. 2019. Neural graph collaborative filtering. In *SIGIR*. 165–174.
[68] Xiang Wang, Tinglin Huang, Dingxian Wang, Yancheng Yuan, Zhenguang Liu, Xiangnan He, and Tat-Seng Chua. 2021. Learning Intents behind Interactions with Knowledge Graph for Recommendation. In *WWW*. ACM / IW3C2, 878–887.
[69] Xiaoqi Wang and Han Wei Shen. 2022. GNNInterpreter: A Probabilistic Generative Model-Level Explanation for Graph Neural Networks. In *International Conference on Learning Representations*.
[70] Xiang Wang, Dingxian Wang, Canran Xu, Xiangnan He, Yixin Cao, and Tat-Seng Chua. 2019. Explainable Reasoning over Knowledge Graphs for Recommendation. In *AAAI*. AAAI Press, 5329–5336.
[71] Ziyang Wang, Wei Wei, Gao Cong, Xiao-Li Li, Xian-Ling Mao, and Minghui Qiu. 2020. Global context enhanced graph neural networks for session-based recommendation. In *SIGIR*. 169–178.
[72] B. Yu. Weisfieler and A. A. Leman. 1968. The reduction of a graph to canonical form and the algebra which appears therein. *NIT* 2 (1968).
[73] Le Wu, Peijie Sun, Yanjie Fu, Richang Hong, Xiting Wang, and Meng Wang. 2019. A neural influence diffusion model for social recommendation. In *SIGIR*. 235–244.
[74] Shiwen Wu, Fei Sun, Wentao Zhang, Xu Xie, and Bin Cui. 2022. Graph neural networks in recommender systems: a survey. *Comput. Surveys* 55, 5 (2022), 1–37.
[75] Shu Wu, Yuyuan Tang, Yanqiao Zhu, Liang Wang, Xing Xie, and Tieniu Tan. 2019. Session-based recommendation with graph neural networks. In *AAAI*, Vol. 33. 346–353.
[76] Shiwen Wu, Wentao Zhang, Fei Sun, and Bin Cui. 2020. Graph Neural Networks in Recommender Systems: A Survey. *CoRR* abs/2011.02260 (2020). arXiv:2011.02260 https://arxiv.org/abs/2011.02260
[77] Yikun Xian, Zuohui Fu, S. Muthukrishnan, Gerard de Melo, and Yongfeng Zhang. 2019. Reinforcement Knowledge Graph Reasoning for Explainable Recommendation. In *SIGIR*. ACM, 285–294.
[78] Fengtong Xiao, Lin Li, Weinan Xu, Jingyu Zhao, Xiaofeng Yang, Jun Lang, and Hao Wang. 2021. DMBGN: Deep Multi-Behavior Graph Networks for Voucher Redemption Rate Prediction. In *SIGKDD*. ACM, 3786–3794.
[79] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How Powerful are Graph Neural Networks?. In *ICLR*.
[80] Makoto Yamada, Wittawat Jitkrittum, Leonid Sigal, Eric P Xing, and Masashi Sugiyama. 2014. High-dimensional feature selection by feature-wise kernelized lasso. *Neural computation* 26, 1 (2014), 185–207.
[81] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. 2018. Graph convolutional neural networks for Web-scale recommender systems. In *SIGKDD*. 974–983.
[82] Zhitao Ying, Dylan Bourgeois, Jiaxuan You, Marinka Zitnik, and Jure Leskovec. 2019. GNNExplainer: Generating Explanations for Graph Neural Networks. In *NeurIPS*. 9240–9251.
[83] Hao Yuan, Jiliang Tang, Xia Hu, and Shuiwang Ji. 2020. XGNN: Towards model-level explanations of graph neural networks. In *SIGKDD*. ACM, 430–438.
[84] Hao Yuan, Haiyang Yu, Shurui Gui, and Shuiwang Ji. 2022. Explainability in graph neural networks: A taxonomic survey. *IEEE transactions on pattern analysis and machine intelligence* 45, 5 (2022), 5782–5799.
[85] Hao Yuan, Haiyang Yu, Jie Wang, Kang Li, and Shuiwang Ji. 2021. On explainability of graph neural networks via subgraph explorations. In *ICML*. PMLR, 12241–12252.
[86] Tong Zhao, Julian McAuley, and Irwin King. 2015. Improving latent factor models via personalized feature projection for one class recommendation. In *CIKM*. 821–830.
[87] Jiawei Zheng, Hao Gu, Chonggang Song, Dandan Lin, Lingling Yi, and Chuan Chen. 2023. Dual Interests-Aligned Graph Auto-Encoders for Cross-domain Recommendation in WeChat. In *CIKM*. ACM, 4988–4994.
[88] Markus Zopf. 2022. 1-WL expressiveness is (almost) all you need. In *International Joint Conference on Neural Networks (IJCNN)*. IEEE, 1–8.