

Why-Not Explainable Graph Recommender

Hervé-Madelein Attolou

ETIS, CY Cergy Paris University, ENSEA, CNRS UMR8051
Cergy-Pontoise, France
herve-madelein.attolou@cyu.fr

Kostas Stefanidis

Tampere University
Tampere, Finland
konstantinos.stefanidis@tuni.fi

Katerina Tzompanaki

ETIS, CY Cergy Paris University, ENSEA, CNRS UMR8051
Cergy-Pontoise, France
Aikaterini.Tzompanaki@cyu.fr

Dimitris Kotzinos

ETIS, CY Cergy Paris University, ENSEA, CNRS UMR8051
Cergy-Pontoise, France
Dimitris.Kotzinos@cyu.fr

ABSTRACT

Explainable Recommendation Systems (RS) enhance the user experience on online platforms by recommending personalized content, as well as explanations for the given recommendations to add transparency and build up trust in the platforms. Extending the notion of explainable RS, in this paper we define Why-Not explanations for recommendations that were expected but not returned, and propose and implement a technique for computing Why-Not explanations in a post-hoc manner for a graph-based RS. Our approach builds on the notion of *counterfactual explanations* in the means of a set of user-rooted edges to add or remove in the graph, in order to place the missing recommendation to the top of the recommendation list, and provides in this way actionable insights on the source data and their interrelations. Our experimental evaluation on a real-world data set demonstrates the feasibility of our proposal and reveals interesting directions for future work.

CCS CONCEPTS

• Information systems → Recommender systems;

KEYWORDS

Explanations; Why-Not questions; Explainable AI; Recommenders

1 INTRODUCTION

Recommendation Systems (RS) allow users to discover personalized content. Traditionally used on media platforms and e-commerce websites, they improve the quality of the browsing experience and increase the impression metrics and conversion rate of the recommended products. The vast quantity of user and product information that companies dispose of is the ‘fuel’ of RS that allow for predicting the items a consumer is most likely interested in, by creating and matching user-item profiles (content-based), by exploiting other users’ feedback as well, e.g., ranking on items (collaborative filtering), or by combining the two aforementioned

categories in hybrid solutions. Moreover, based on the modeling of the available information (for example, in matrices or tables) different kinds of RS can be proposed. In this paper, we focus on *graph-based RS* that represent the information on items, users, their characteristics, and their relations in the form of a graph.

Unexpected (either existing or missing) and/or unjustified recommendations may frustrate the users and can be detrimental to their trust and loyalty to the system. Particularly explaining existing recommendations (and in general machine learning results) has been an active scientific problem for at least the last decade. To add transparency, *Explainable RS* share a piece of information from the system with users, so that they understand how/why the recommendation was computed. This information is commonly called an *explanation* and can either be system-agnostic and data-specific information (e.g., user/item characteristics, previous actions), or system-specific information (e.g., a set of rules in a rule-based decision system).

Whereas a lot of attention has been given in the literature for explaining existing recommendations, in this paper we focus on the *missing recommendation* explanation problem. Thus, we aim to provide an explanation to the user for why some expected, user-defined item is not recommended at the top of the list. In various scenarios such items are known to the users, e.g., during system testing/debugging/verification or for diversity enhancement, as assumed also in the line of works on Why-Not questions in different domains (see Section 2).

The type of information to render as an explanation depends on the expertise and the purpose of the system user, who can be the end-user of the platform (‘naive’ user) or the system designer (expert user). For end-users, explanations serve as a means to increase their trust in the platform by providing hints on the actions or characteristics that fired the recommendations. For the system designers trying to understand their data and their system, and eventually remodel/debug the system, explanations can also expose deeper system parameters or data relations.

In this paper, we target the end-user or the system developer who explores the system data and their relationships, possibly for debugging purposes, and who wants to understand which of their actions were the causes of missing specific recommendations, or what actions they should have performed in order to receive the missing recommendations. We choose to return only the current user’s actions in the explanations for data privacy reasons, however, this is an implementation choice that can be altered in the convenience of the system. In this way, we opt for a form of *Counterfactual*

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ICDE 2024, April 16th - April 19th, 2024, Utrecht, Netherlands

© 2024 Association for Computing Machinery.

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...\$15.00

<https://doi.org/10.1145/nnnnnnnn.nnnnnnnn>

Explanations (CFE), i.e., proposing a possible world that could have led to the desired outcome. Figure 1 demonstrates our motivating example. Paul, the target user at node 2, receives ‘Python’ as a recommendation, however they are interested to know why ‘Harry Potter’ is not recommended, so they ask “Why not Harry Potter?” Figure 1a provides the Why-Not explanation composed by two *past actions of the user*, and can be read as “Had you not interacted with Candide and C, your top recommendation would be Harry Potter”. Figure 1b provides the Why-Not explanation composed by one *possible action* that the user can perform, and can be read as “Had you interacted with Lord of the Rings, your top recommendation would be Harry Potter”.

As previously said, related work in the literature has addressed different versions of the *Why Problem*. However we argue that explaining why an item *WNI* has not been top-recommended to a user is a different problem from explaining a *Why* interpretation of the same question like: ‘Why is *WNI* in the *k*th position?’, ‘Why is the current top-1 in the first position?’, or even ‘Why is the ranking list up to *k* as it is?’. The problems are related, however they are not equivalent neither are they practically inclusive the one of the other. Take for instance the second case, when someone could argue that the Why explanation for the top-1 item is the explanation for the *WNI* not being recommended. A Why-explanation algorithm (for instance [11]) would yield the CFE as seen in Figure 2: “Had you not read C you would have obtained The Alchemist as a recommendation.” This is a different explanation than the one proposed in Figure 1a and does not yield the *WNI* in the top-1 position. Thus, our solution addresses a different problem than traditional Why explanations, them being counterfactual or not, and needs to be addressed with a different solution.

In this work, we propose *EMiGrE*, a framework for providing Why-Not explanations for graph RS. The solution relies on building Counterfactual Explanations grounded in past or suggested user actions.

More concretely, our contributions are listed as follows:

- We introduce the problem of Why-Not questions in graph RS.
- We define *causal* Why-Not explanations as a set of existing or missing (user-rooted) edges from the graph, inspired by Counterfactual Explanations.
- We propose two modes (adding/removing edge mode) to compute Why-Not Counterfactual Explanations and a number of heuristics for small or fast explanations.
- We run a thorough, comparative experimental evaluation analysis of our proposal in a large, real-world dataset. The results show the feasibility of our solution and the quality of the proposed explanations in adapted scenarios.

EMiGrE is built on the popular Personalised Page Rank algorithm [38], but can be adapted to other user-defined functions that compute the importance of a node for another node monotonically in the number of edges that connect the two nodes.

2 RELATED WORK

The subject of Why-Not Questions is a recent notion in the field of explainable artificial intelligence (XAI) and more precisely in explainable recommenders, however, it has been studied in other

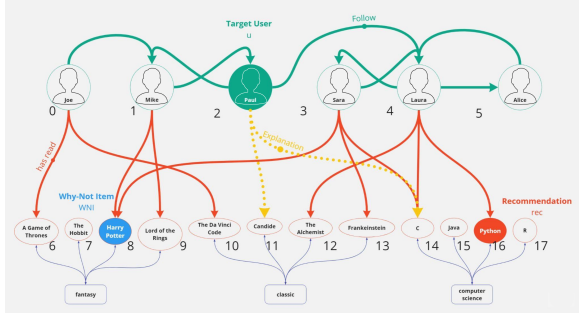
contexts, such as relational database queries. Thus, in this section, we discuss related work on two relevant subjects: (i) Recommenders and Explainable Recommenders, and (ii) Why-Not Questions and Explanations. Due to the lack of space, we do not mention here general XAI literature.

2.1 Recommenders

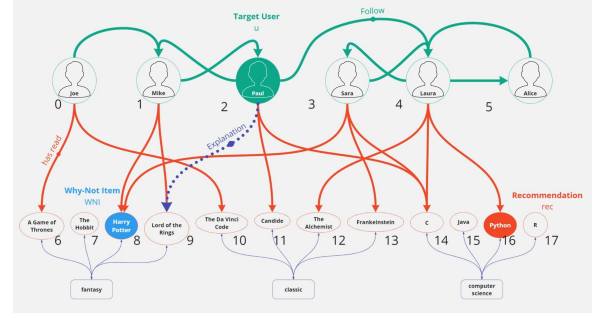
One way to build a recommendation list is to use content-based recommenders (see [19] for a survey). These algorithms use the similarities between the features of the items and the users’ profiles to make recommendations. They require complex semantic analysis of the content and the extraction of the users’ preferences to match them with their expected next interest.

Score-based (or collaborative filtering) recommenders [29] exploit ratings and other quantifiable user-system interactions given to the items by the users. Modeling the preferences of users as precise ratings allows a representation as a matrix and the use of matrix-based computations. Direct feedback from the user ensures a better connection between the recommendations and past activity. Some implementations include Non-Negative Factorization [15], Singular value decomposition SVD [14], SLIM [25] methods inspired by item-KNN with different optimization. Recent implementations include Graph Neural Network Models like NGCF [34] by modelizing the User-Item relations into an embedding graph. Context-based recommenders exploit information related to the situation that the items and/or the users are in, like for example their spatiotemporal configuration [31, 36].

Explainable Recommenders. Explainable Recommenders accompany the recommendations with either inherent to the model explanations or post-processing explanations.. The approach to explain a system depends on whether the system is *white*, i.e., exposing the information about its internal workings, or *black*, i.e., only the input and the output are known (see [40] for a recent survey). In the case of black-box recommenders, the explanations consist in interpreting the results by revealing relationships in the input data [16], or by pinpointing the importance of the different features that contribute to the predicted value [20]. In the case of a white box system, the explanations consist of dwelling on the intrinsic characteristics of the RS to truly explain the system [11]. Deep Learning methods, such as CNN (Convolutional Neural Network) by, for example, relying on review text [30], RNN (Recurrent Neural Network) [10] or more recently memory networks [33], allowing visually explainable recommendation highlighting the image region of interest for a user. Collaborative filtering is among the methods that provide a solution to this problem by leveraging data from ratings [27], user opinion [21], summarized topics [35]. Other methods include: Rule mining for textual explanations [1] and Post-hoc processing [8]. User surveys demonstrate the quality of the provided explanations in terms of transparency and also, express the need for explanations in automated recommendation systems [12]. In [2], the authors propose an explainability-constrained Matrix Factorisation technique that computes the top-n recommendation list from items that are explainable. Here, the *explainability* is an additional constraint in the Collaborative Filtering model on top of the others. This method is effective in generating accurate and explainable recommendations with explanations taking the form of



(a) Why-Not explanation with $A^- = \{(2, 11), (2, 14)\}$ (links in dotted yellow).



(b) Why-Not explanation with $A^+ = \{(2, 9)\}$ (blue dotted link).

Figure 1: A graph book recommendation system with user-user edges (in green), user-book edges (in red), and book-category edges (in blue). Paul (2) is recommended *Python* (node in red) (16) and asks "Why-Not *Harry Potter* (node in blue)?"

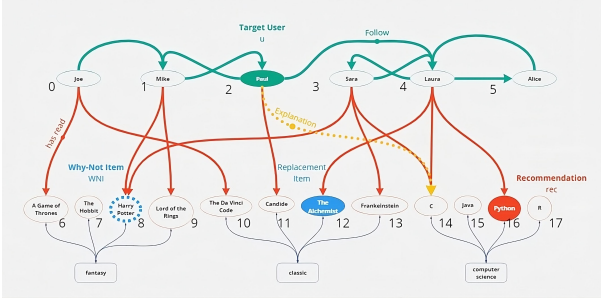


Figure 2: CFE from PRINCE with $A^* = \{(2, 14)\}$ (links in dotted yellow).

neighbor-style explanations (either user-based or item-based) but is less actionable, for example, than Counterfactual Explanations (CFE). Furthermore, data modeling limits the performance in more diverse and complex scenarios. Our methods address this issue by using HIN (see Section 3.1) to describe diverse data types and relationships.

2.2 Why-Not Questions and Explanations

Explaining and debugging queries in the absence of expected results has been studied in various domains like traditional databases [3, 4], top-k spatial queries [16], workflow analysis [6], regular expressions [28], streaming-data applications [26] and in SPARQL [37]. It is provided by means of instance-based (existing/missing input data points/tuples), query-based (faulty operators/data manipulations), or modification-based (modified queries/settings) explanations. In [32], the authors defined the problem of Why-Not questions for collaborative filtering RS and proposed an algorithm for producing Why-Not explanations tailored to the system developer. Recently, in [7] the authors consider Why-Not Questions in the context of top-k queries and score-based ranking functions. This could be adapted to an RS scenario but would not fit the same use case as our work, as it considers the item as a vector of features that could be adjusted to answer the Why-Not Question. Here we target modification of the actions of the user in a graph data scenario.

We can find in the literature the term *Contrastive explanation* [18, 22, 23]. Miller [22] surveyed philosophy, psychology, and cognitive science literature on the methods people employ to explain outcomes to each other. The conclusion describes that people would rather ask "Why P rather than Q?" than "Why P?". [18] refers to this type of question as *contrastive*, P as the *fact* and Q as the *contrast case* or as the *foil*. Contrastive questions (and explanations) are different from our Why-Not questions (and explanations); the formulation of the question differs, as the Why-Not question is more general. We ask "Why not Q?" and not "Why P rather than Q?". This implies many more possibilities for the explanations, as the answers should not only reason about the original fact P versus the foil Q, but also verify the conditions under which the explanation is true w.r.t. any other foil. As a result, we need to propose a new formal setting for Why-Not explanations and devise new techniques for computing them.

3 PRELIMINARIES

Recommenders allow end-users to receive information selected from automatically tuned criteria. We differentiate multiple categories of recommenders based on their abilities and methods.

Formally, we have a set of items I and a set of users U , where each user provides ratings for a subset of I . Specifically, a user $u \in U$ shares their preferences (rating, buying, watching, etc.) for item $i \in I$ (optionally with a score s). The subset of items that are possible to recommend to a user is described as the set of target items $T \subset I$. For every target item $t \in T$ not rated by a user u , the recommender estimates a relevance score, $p(u, t)$. The item with the highest score designates the recommended item rec . The items with a high relevance score for u will compose the recommendation list for the user in case of a *top-n Recommender*.

Definition 3.1 (Heterogeneous Information Networks (HIN) [11]). A heterogeneous graph $G = (V, E, \theta)$ consists of a set of nodes V , a set of edges $E \subseteq V \times V$, and a mapping θ from each node and each edge to their types, such that $\theta_V : V \rightarrow T_V$ and $\theta_E : E \rightarrow T_E$ with $|T_V| + |T_E| > 2$. A directed and weighted heterogeneous graph, where each node $v \in V$ and each edge $e \in E$ belong to exactly one type is called a *Heterogeneous Information Network (HIN)*.

In our setting, we consider a graph with at least two node types, (i) users U , and (ii) items I . More node types and weights can be created, expressing more complex information from the dataset. The graph is directed with the direction being a choice motivated by the definition of the action/information modeled by the edge. For instance, social interaction like the *follows* relationship is not always reciprocal, thus the edge denoting this action should be directed from the follower to the followed.

3.1 Counterfactual Explanations

de Graaf and Malle [9] claim that because people assign human-like traits to artificial agents, people will expect explanations using the same principle used to explain human behaviors. In the case of Explainable Recommenders, a form of human-like explanation [23] can be a Counterfactual Explanation (CFE). We can define it as a set of actions responsible for the provided recommendation. Alternatively, we can say that, if the user have not performed this set of actions, the recommendation would have been different. In a HIN, altering the graph by removing the elements in the CFE would lead to a different item being selected as a recommendation.

Definition 3.2 (Counterfactual Explanation [11]). With u the target user, A_u the set of all actions of u and r_u the item recommended to u , the CFE for u is $A^* \subseteq \{(u, n_i) | (u, n_i) \in A\}$ where n_i is a neighbor of u , such as using $A - A^*$ as an input of the recommenders lead to $r_{u^*} \neq r_u$ being the recommendation. A minimal counterfactual explanation is the smallest set of actions satisfying the property.

This form of explanation provides user-comprehensible and actionable evidence of the trustworthiness of the system. Restricting the solution space to the neighborhood of the user node also set a privacy preserving constraint as only information already available to the user is disclosed. An example of HIN describing a book recommendation system can be seen in Figure 1a.

3.2 Personalized PageRank for Explainable Recommenders

To compute the relation between the users and the candidate item, we use Personalized PageRank (PPR) [13] for recommendation [24] in HINs. PPR is the stationary distribution of a random walk in G in which, at a given step, with probability α , a surfer teleports to a set of seed nodes $\{s\}$, and with probability $1 - \alpha$, continues the walk to a randomly chosen outgoing edge from the current node. In the graph G , teleportation probability α , a single seed s , the one-hot vector e_s , and the transition matrix W , the Personalized PageRank vector $PPR(s)$ is defined recursively as:

$$PPR(s, \cdot) = \alpha e_s + (1 - \alpha) PPR(s, \cdot) W \quad (1)$$

with $PPR(s, v)$ be the PPR score of node v personalized for the user s . We get rec the PPR recommendation for user $u \in U$ with:

$$rec = \arg \max_{i \in I \setminus N_{out}(u)} PPR(u, i) \quad (2)$$

While at the center of our work, the computation of the PPR is not an easy or lightweight task, it involves a lot of time and power. Two algorithms allow us to maintain an approximation of this metric: Forward Local Push (FLP) and Reverse Local Push

(RLP)[39]. Both are based on the recursive exploration of the graph but use different starting points and present some key advantages depending on the use case.

The first, FLP, is starting the exploration from the source node (in our case the target user node) and updating the probability along the neighbor nodes through the outgoing edges.

Alternatively, RLP initiates the exploration from a candidate node (item to be ranked in the recommendation list) and pushes the values back to the neighbor nodes through the in-going edges. Both versions use an estimate $\vec{P}(s, t)$ of $PPR(s, t)$ for each target $t \in V$ and a residual $\vec{R}(s, t)$ for s .

Let $\vec{P}(t) = \vec{P}(\cdot, t)$ denote the vector of estimates and $\vec{R}(t) = \vec{R}(\cdot, t)$ for the vector of residuals, with the arrow denoting the direction of the update (from the source along the neighbors). In the FLP case:

$$PPR(s, t) = \vec{P}(s, t) + \sum_{x \in V} \vec{R}(s, x) \times PPR(x, t), \forall s \in V \quad (3)$$

On the other hand, estimates and residuals satisfy an invariant property in the RLP case in the opposite direction:

$$PPR(s, t) = \vec{P}(s, t) + \sum_{x \in V} PPR(s, x) \times \vec{R}(x, t), \forall s \in V \quad (4)$$

With $\vec{P}(t) = \vec{P}(\cdot, t)$ denote the vector of estimates and $\vec{R}(t) = \vec{R}(\cdot, t)$ for the vector of residuals. Again the arrow denotes the direction of the update (from the item nodes to the source node).

PRINCE, Provider-side Interpretability with Counterfactual Evidence [11] is a model that relies on a graph modelization of the data to provide a counterfactual explanation of the recommendation to the user. PRINCE provides a minimal CFE that describes the smallest set of actions that needs to be undone to change the recommendation to *any* other replacement item. This algorithm places itself as a post-processing phase of the RecWalk framework [24], which is a state-of-the-art system based on the Personalized PageRank (PPR) computation. By altering the actions performed by the user, PRINCE can lower the PPR of the recommended item (r_u) to advantage the PPR of other items, one of which becomes the replacement item (r_{u^*}). PRINCE is effective at providing consumer-centered Counterfactual Explanations regarding some privacy constraints, but generally cannot answer Why-Not questions. Take for instance our running example and the Why-Not explanation illustrated in Figure 1a. If we try to answer the same Why-Not question by means of a CFE on the top-recommended item provided by PRINCE, we obtain the result illustrated in Figure 2; here the explanation is different and leads to a different replacement item (*The Alchemist*) and not to our Why-Not item (*Harry Potter*). In summary, our work differs from PRINCE in that:

- (1) We provide explanations for missing recommendations, whereas PRINCE provides explanations for existing recommendations.
- (2) We provide more actionable explanations, by proposing in the explanations not only existing actions (as in PRINCE), but also not yet existing actions (Section 5.1).

4 PROBLEM DEFINITION

In this paper, we want to provide the means for a user to understand the recommendation system, by explaining the absence of

an interesting -to the user- item from his/her recommendation list. We term such a missing recommendation as a *Why-Not question* (or *Why-Not item*), and we further define it as an item that is not ranked at the top of the recommendation list and the user has not interacted with yet (Definition 4.1). In general, Why-Not questions can be expressed in different granularities: one item, a set of items, or a category of items. In this paper, we consider only a single item as the Why-Not question and leave the other classes (that could include categories or a selected group of items) as future work. Note also that in what follows, for brevity reasons, when we refer to the graph G we may imply the graph recommendation system as well.

Definition 4.1. Given a HIN $G = (V, E, \theta)$, the set of items $I \subset V$, a user $u \in U$, $U \subset V$, and its initial recommendation rec , a *Why-Not question* is an item $WNI \in I \setminus \{rec\}$ such that $(u, WNI) \notin E$.

Once a Why-Not question is defined by the user, the purpose of this work is to provide an explanation for the absence of WNI from the top recommendation position. We define the *Why-Not Explanation* as the set of edges rooted at the user u node, which can either be added or removed from the HIN to replace rec by WNI as the recommendation. Contrary to the CFE defined in PRINCE (see Definition 3.2), in the missing recommendation setting the cause for the absence of a recommendation may not be explained only (or better) by pre-existing actions of the user; indeed the user may miss actions that can lead to pertinent recommendations. Furthermore, we impose an important constraint in the definition of our Why-Not explanation; the replacement item r_u^* must be exactly the item WNI . In other words, Definition 3.2 explains a recommendation with the foil being *any other item*, while in the Why-Not explanation we need to explain the recommendation using the Why-Not item WNI as a foil. This constraint has implications in the complexity of the computations, as we will see later in Section 5. We precisely expect the explanation to replace the recommendation rather than simply increase its ranking in the recommendation list as it is.

Definition 4.2. Given a HIN $G = (V, E, \theta)$, a user u , and a Why-Not Item WNI , and the recommendation rec , a Why-Not explanation for WNI is the set of edges $A^* \subseteq A$, $A \in \{A^+, A^-\}$, with $A^+ = \{a^+ | a^+ = (u, i) \notin E, i \in I\}$ and $A^- = \{a^- | a^- = (u, i) \in E, i \in I\}$ such that $G' = (V, E', \theta)$ with $E' = E \cup A^+$ or $E' = E \setminus A^-$ generates WNI as the top-1 recommendation instead of rec .

Figure 1 describes an example graph book recommendation system, where a user, Paul, is recommended Python. Paul wants to know why he is not recommended Harry Potter. Based on Definition 4.2, we can have two kinds of Why-Not explanations. Figure 1a shows a Why-Not explanation composed by the edges (2, 11) and (2, 14) to be removed from the graph, and which reads as: "If Paul had not read *Candide* (11) and *C* (14) (links in yellow), the recommendation would be *Harry Potter* (8)". Similarly, Figure 1b shows a Why-Not explanation composed by the edges (2, 9) to be added to the graph, and which reads as: "If Paul had read the book *The Lord of the Rings* (9) (blue dotted link), then the recommendation would be *Harry Potter* (8)".

5 THE EMIGRE ALGORITHM

In this section, we develop *EMiGRe*, an algorithm for why an item is not recommended in the case of a graph-based recommendation

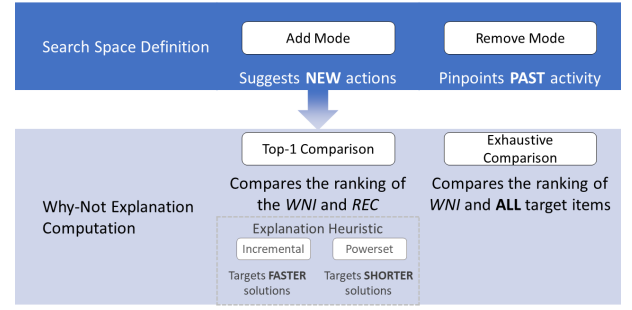


Figure 3: The *EMiGRe* framework.

system, described in Figure 3. Based on Definition 4.2, the Why-Not explanations consist of a set of edges, a.k.a, actions from the user, which should be added or removed from the graph in order to change the top-1 recommendation in favor of the WNI item. Thus, we devise two modes of the algorithm: (i) the *Add mode* for computing explanations consisting in missing pertinent edges to be added in the graph, and (ii) the *Remove mode* for pinpointing the existing edges to be removed from the graph (Section 5.1).

Then, we propose two methods for selecting the best candidate edges, either comparing the ranking of WNI only with the ranking of the top-1 recommendation or exhaustively comparing with the ranking of all the items (Section 5.2). Finally, in the top-1 comparison case, we propose two heuristics (Section 5.2.1); the *Incremental* one focuses on run time (Section 5.2.1), providing faster solutions, while the *Powerset* one focuses on the size of the explanations, benefiting smaller explanations (Section 25).

5.1 Search Space Definition

The main idea in the quest of candidate neighbors for the explanations is to use the PPR (see Section 3.2) to find which edges are the most 'influential' for WNI . Our choice of PPR is grounded by the popularity of this algorithm for graph recommendation systems. However, our Why-Not explanation definition is not tight to the type of graph recommender. Moreover, our solution in *EMiGRe* can be adapted to other user-defined functions that compute the importance of a node i for another node j monotonically in the number of edges rooted on j and that connect the two nodes.

In what follows, we specify the process in the *Remove Mode*, where we search among already existing edges, and the *Add Mode*, where we search among non-existing edges.

5.1.1 Remove Mode. Algorithm 1, Line 4 is going through A , the list of the user's u output edges of an acceptable type defined in T_e . The corresponding neighbors (n_i) are ordered by their relative contribution to the recommendation as described in Equation 5, and stored in the automatically sorted list H .

Equation 5 measures the relative contribution (influence) of the item n_i on WNI taking into consideration the existence of the item rec and weighted by the link's (u, n_i) weight, given in the matrix W . If the contribution is positive, the neighbor n_i contributes more to the PPR of the currently recommended item rec than to the PPR of WNI , which means that its removal from the graph will benefit

WNI. A negative contribution implies that n_i contributes more to the *PPR* of *WNI* than to the *PPR* of *rec*, thus its removal will not influence positively *WNI*.

We also compute τ that corresponds to the sum of the contributions of all neighbors added in H . This serves as a threshold for the next steps (more precisely in Algorithm 4, Line 24 and Algorithm 3, Line 16). At the end of Algorithm 1, τ will be positive because in the current setting *rec* dominates *WNI*. Subsequent changes in the graph, in the quest of a Why-Not explanation will make it negative, which indicates a change in the ranking of *rec* and *WNI*, and thus a possible Why-Not explanation.

$$\text{contribution}_{rmv}(n_i) = W(u, n_i) \cdot (\text{PPR}(n_i, \text{rec} \mid A) - \text{PPR}(n_i, \text{WNI} \mid A)) \quad (5)$$

Algorithm 1: Search Space Definition - Remove Mode.

Data: $G = (V, E, \theta), I \in V, u \in V, \text{rec} \in I, \text{WNI} \in I, T_e \subseteq T_E$
Result: H (the list of nodes for the explanation), τ

```

1  $A \leftarrow \{(u, n_i) \mid n_i \in N_{out}(u), n_i \neq u, \theta_E((u, n_i)) \in T_e\};$ 
2  $H \leftarrow \text{DescendingOrderList}(\emptyset);$ 
3  $\tau \leftarrow 0;$ 
4 foreach  $(u, n_i) \in A$  do
5    $\text{contribution} \leftarrow \text{contribution}_{rmv}(n_i);$ 
6    $H.\text{insert}(n_i, \text{contribution});$ 
7    $\tau \leftarrow \tau - \text{contribution};$ 
8 end
9 return  $H, \tau;$ 
```

5.1.2 Add Mode. In Algorithm 2, after some initialisation steps, Line 6 computes the threshold τ , which is necessary for the later phases of the process (see Section 5.2). Using the *Reverse Local Push* algorithm (Line 8) from [39] and the Why-Not Item *WNI* as the source node, we get the list PPR_{WNI} of potential neighbors together with their computed *PPR* values, filtered by their edge type so that it exists only in T_e . Line 10 computes for each potential neighbor n_i in PPR_{WNI} its relative contribution to *WNI* as described in Equation 6. Note that Equation 6 is similar to Equation 5, but we omit the matrix W because non-existing edges do not have any weights. Moreover, now a positive contribution means that n_i influences more *WNI* than *rec*, so its addition to the graph would benefit *WNI*. Finally, the potential neighbors are stored in the ordered-by-contribution list H .

$$\text{contribution}_{add}(n_i) = \text{PPR}(n_i, \text{WNI} \mid A) - (\text{PPR}(n_i, \text{rec} \mid A)) \quad (6)$$

5.2 Why-Not Explanation Computation

Either from the Remove mode or the Add mode, the search space definition phase of *EMiGRe* provides a list of nodes along with their contributions to *WNI*. The addition or deletion (depending on the mode) of the edge connecting each node and u to/from the graph rearranges the items in the recommendation list. The final purpose of *EMiGRe* is to propose the deletions or additions that will allow the placement of *WNI* at the top of the recommendation

Algorithm 2: Search Space Definition - Add Mode.

Data: $G = (V, E, \theta), I \in V, u \in V, \text{rec} \in I, \text{WNI} \in I, T_e \subseteq T_E$
Result: H (the list of nodes for the explanation), τ

```

1  $A \leftarrow \{(u, n_i) \mid n_i \in N_{out}(u), n_i \neq u, \theta_E((u, n_i)) \in T_e\};$ 
2  $H \leftarrow \text{DescendingOrderList}(\emptyset);$ 
3  $\tau \leftarrow 0;$ 
4 foreach  $(u, n_i) \in A$  do
5    $\text{contribution} \leftarrow \text{contribution}_{rmv}(n_i);$ 
6    $\tau \leftarrow \tau - \text{contribution};$ 
7 end
8  $\text{PPR}_{\text{WNI}} \leftarrow \text{ReverseLocalPush}(\text{WNI}, T_e)$ 
9 foreach  $n_i \in \text{PPR}_{\text{WNI}}$  do
10   $\text{contribution} \leftarrow \text{contribution}_{add}(n_i);$ 
11   $H.\text{insert}(n_i, \text{contribution});$ 
12 end
13 return  $H, \tau;$ 
```

list and which will form the explanation. We further propose two alternatives for verifying that a candidate explanation is correct; the top-1 and the *Exhaustive Comparison*.

5.2.1 Top-1 Comparison. In the top-1 comparison alternative, for each new possible graph, i.e., after adding (deleting) an edge from u to (from) $n_i \in H$, we compare the importance of *WNI* - measured by its *PPR* - only to the original top-1 recommendation (the *rec* item). This allows us to see if, in this new graph, *WNI* will overtake *rec*.

Moreover, the way we consider the nodes of the list H allows us to optimize the process either on time or in explanation size. Thus, we present in the following the *Incremental* heuristic, focusing on time, and the *Powerset* heuristic, focusing on size. While *Incremental* goes through a solution sub-space by increasing the size of the explanation at each iteration, *Powerset* processes all possible pertinent edge combinations in an ascending combination size manner to favor smaller solutions.

Incremental. Our *Incremental* heuristic removes/adds one by one edges from the nodes of H to u , until it finds a Why-Not explanation.

Algorithm 3 describes the process in detail. Following one of the algorithms described in Section 5.1, it receives input H , the nodes that can be used to build the explanation, and the threshold τ . The specificity of Algorithm 3 lies in the fact that at every iteration, we incrementally add/remove the next most influential node, i.e., the next one with the highest contribution in H (Line 5). At each pass, we also update the threshold by adding/subtracting the node's contribution from the threshold. When the threshold reaches 0 (i.e., the point up to which n_i favors *WNI* over *rec* at Line 16), we check if indeed in the new graph with the edges removed/added, *WNI* is the top-1 recommendation (Lines 18).

The function "TEST" at Line 18 refer to a function that tests if the provided set of edges is a Why-Not explanation. This same function will be used in other Algorithms

Powerset. In the *Powerset* heuristic (Algorithm 4), we create the power set of H , where each combination's contribution is the sum of the contributions of the individual nodes participating in the

Algorithm 3: Incremental Why-Not explanation computation.

Data: $G = (V, E, \theta), I \in V, u \in V, rec \in I, WNI \in I, H$ (the list of nodes for the explanation), $\tau, MODE$

Result: A^* (the explanation for u and WNI)

```

1  $A \leftarrow \{(u, n_i) \mid n_i \in N_{out}(u), n_i \neq u\};$ 
2  $A^* \leftarrow \emptyset;$ 
3  $check \leftarrow False;$ 
4 while  $|H| > 0$  and  $check == False$  do
5    $(n_i, contribution) \leftarrow H.pop(0);$ 
6   /* we prune the negative contribution from the list*/
7   if  $contribution > 0$  then
8     if  $MODE == ADD$  then
9        $\tau \leftarrow \tau + contribution;$ 
10       $A^* \leftarrow A^* \cup \{(u, n_i)\};$ 
11    end
12    if  $MODE == REMOVE$  then
13       $\tau \leftarrow \tau - contribution;$ 
14       $A^* \leftarrow A^* \cup \{(u, n_i)\};$ 
15    end
16  end
17  if  $\tau > 0$  then
18     $check \leftarrow test(G, A^*, MODE);$ 
19  end
20 end
21 if  $check == True$  then
22   return  $A^*;$ 
23 else
24   return  $\emptyset;$ 
25 end

```

combination (Line 8). We then iterate through all the combinations by ascending size and contribution value (if tie), and perform the same kind of checks as in Algorithm 3.

5.2.2 Exhaustive Comparison. Up to now, the different alternatives were making use of the contribution in Equation 6 or 5, which indicates whether a node is contributing more to the current recommendation rec or to the Why-Not item WNI . However, we also needed to ensure that WNI will not be dominated by any other (target) item, verified by the final check in Algorithms 3 and 4.

To address this issue, we propose to incorporate a more nuanced monitoring system that takes into account not just the sign of the contribution value, but also its magnitude. By doing so, we can identify nodes that may be contributing more to WNI than to rec , but also may favor more another item t , which may dominate WNI in the final recommendation list. Then, we avoid considering such items as candidates for the explanation.

Algorithm 5 first computes the contribution $C_{n,t}$ of $n \in H$ to WNI , relative to each item t in the recommendation list (Lines 7 and 14).

To select the set of neighbors to be removed or added to the graph, we apply a condition to the weights of the matrix. This condition ensures that WNI is the top item. We also need to consider the

Algorithm 4: Powerset Why-Not explanation computation.

Data: $G = (V, E, \theta), I \in V, u \in V, rec \in I, WNI \in I, H$ (the list of nodes for the explanation), $\tau, MODE$

Result: A^* (the explanation for u and WNI)

```

1  $A \leftarrow \{(u, n_i) \mid n_i \in N_{out}(u), n_i \neq u\};$ 
2  $A^* \leftarrow \emptyset;$ 
3 /* we prune the irrelevant contributions from the list*/
4 foreach  $(n_i, contribution) \in H$  do
5   if  $contribution \leq 0$  then
6      $H.remove(n_i, contribution)$ 
7   end
8  $HP \leftarrow powerset(H);$ 
9 foreach  $c \in [0, |HP|]$  do
10   $h_c \leftarrow$  all combinations of size  $c$  ordered by contribution;
11   $\tau_c \leftarrow \tau;$ 
12   $check \leftarrow False;$ 
13  while  $|h_c| > 0$  AND  $check == False$  do
14     $(n_i, contribution) \leftarrow h_c.pop(0);$ 
15     $A^* \leftarrow A^* \cup \{(u, n_i)\};$ 
16    if  $MODE == ADD$  then
17       $\tau_c \leftarrow \tau_c + contribution;$ 
18       $A_{check} \leftarrow A \cup A^*;$ 
19    end
20    if  $MODE == REMOVE$  then
21       $\tau_c \leftarrow \tau_c - contribution;$ 
22       $A_{check} \leftarrow A \setminus A^*;$ 
23    end
24    if  $\tau_c > 0$  then
25       $check \leftarrow test(G, A^*, MODE)$ 
26    end
27  end
28  if  $check == True$  then
29    return  $A^*;$ 
30  end
31 end
32 return  $\emptyset;$ 

```

alternative situation, where WNI is not the top recommendation. To do this, we compute a threshold above which any item from T , the list of target nodes, switches position with WNI .

For each target item t , we compute (at Line 19) the switching threshold as follows:

$$Threshold(t) = \sum_{node}^{N_{out}} C_{node,t} \quad (7)$$

Similar to the Algorithm 2, in Add Mode, the threshold is always computed from the contributions of the nodes in A . For readability purposes, in Algorithm 5 at Line 19, we consider that these necessary values are already computed.

The contribution is now a matrix of size $|H| \times T$ instead of a vector of size $|H|$. Differently than in the previous cases, no filter is applied beforehand based on the sign of the contribution. We keep

Algorithm 5: Why-Not explanation computation using Exhaustive Comparison.

Data: $G = (V, E, \theta), I \in V, u \in V, rec \in I, WNI \in I, H$ (the list of nodes for the explanation), $\tau, MODE, T \subset I$ (the set of target nodes)

Result: A^* (the explanation for u and WNI)

```

1  $A \leftarrow \{(u, n_i) \mid n_i \in N_{out}(u), n_i \neq u\};$ 
2  $A^* \leftarrow \emptyset;$ 
3 /*  $C_{n,t}$  is an entry of  $C$ , the matrix of contributions */;
4 if  $MODE == REMOVE$  then
5   foreach  $n \in H$  do
6     foreach  $t \in T$  do
7        $C_{n,t} \leftarrow$ 
8          $W(u, n) * (PPR(n, t|A) - PPR(n, WNI|A));$ 
9     end
10  end
11 if  $MODE == ADD$  then
12   foreach  $n \in H$  do
13     foreach  $t \in T$  do
14        $C_{n,t} \leftarrow (PPR(n, t|A) - PPR(n, WNI|A));$ 
15     end
16   end
17 end
18 foreach  $t \in T$  do
19    $Threshold_t \leftarrow \sum_n^{N_{out}} C_{n,t};$ 
20 end
21 /* Fill in the lines of  $C$ , for combinations of size  $>1$  */
22 foreach  $l \in [2, |H|]$  do
23    $h_l \leftarrow$  all combinations of size  $l$ ;
24   foreach  $combi \in h_l$  do
25     foreach  $t \in T$  do
26        $C_{combi,t} \leftarrow \sum_j^{combi} C_{j,t};$ 
27     end
28      $C_{combi} \leftarrow C_{combi} - Threshold;$ 
29     /* we select the lines in  $C$  that strictly contain
30       positive values */
31     if  $c_{combi,t} > 0, \forall c_{combi,t} \in C_{combi}$  then
32        $candidates \leftarrow C_{combi};$ 
33     end
34   end
35    $A^* \leftarrow \emptyset;$ 
36   foreach  $n \in combi$  do
37      $A^* \leftarrow A^* \cup \{(u, n)\};$ 
38   end
39    $check \leftarrow test(G, A^*, MODE);$ 
40   if  $check == True$  then
41     return  $A^*;$ 
42   end
43 end
44 return  $\emptyset;$ 

```

all available elements in H to compute the possible combinations. This allows elements that would have been considered irrelevant in the previous cases to be considered here if they can participate in lowering the influence of any item that is not WNI .

Applying the threshold to the matrix entails two constraints; First, it ensures that the weight of all better items decreases by more than the threshold so that the WNI overtakes them; and second, it ensures that the degree of all worse items does not increase by more than the threshold. This is achieved by subtracting the threshold vector from every line of the Matrix C and preserving the positive ones (see Line 29), after extrapolating the contribution values to all possible combinations at Line 21. We still have to perform a check of the potential solutions to remove false positives, at Line 34. This process is justified by the experiment (Section 6) where we compare our solution to the Algorithm without the "CHECK" (i.e. direct).

Example. To clarify Algorithm 5 we use the running example, seen on Figure 1a. For a reminder, we apply the Remove Mode for the source node $u=2$ and the Why-Not Item $WNI=8$ and items in the recommendation list $T=\{6, 7, 9, 8, 10, 13, 17, 15, 12, 16\}$. For this setting, we get an initial contribution matrix C described in Table 1.

Table 1 shows the contribution from each neighbor in N_{out} to each target item t relative to WNI . In this example, $N_{out}=\{1, 5, 11, 14\}$, $T=\{6, 7, 9, 10, 13, 17, 15, 12, 16\}$ (the node 8 is excluded because it's the Why-Not Item WNI). The threshold vector is described in Table 2. As we expected, all items ranked worse than WNI have a negative threshold and the better item has a positive threshold. This means we can decrease the PPR of the target items as much as the threshold indicates.

We then build a contribution matrix (Table 3), for all combinations of neighbors by summing for each combination the individual contributions and subtracting the threshold vector from each contribution line ($C_{combi} - threshold$).

The potential solutions are the combinations that have a vector of positive coordinates and are selected in order of their sizes. Here, the potential solutions to consider are $\{(11, 14), (5, 11, 14)\}$. After the "TEST" phase, we can see that the removal of the edges $(2, 14), (2, 11)$ leads to the $WNI = 12$ becoming the recommendation. Thus, $A^- = A^* = \{(2, 14), (2, 11)\}$.

Table 1: Initial Contribution Matrix

	6	7	9	10	13	17	15	12	16
1	-0.011	-0.011	-0.0	-0.024	-0.024	-0.024	-0.024	-0.024	-0.024
5	-0.002	-0.002	-0.002	0.003	0.005	0.008	0.008	0.011	0.016
11	0.0	0.0	0.0	0.024	0.024	0.0	0.0	0.024	0.0
14	0.0	0.0	0.0	0.0	0.0	0.024	0.024	0.0	0.024

5.3 Complexity and Correctness

The complexity of our solution can be considered with respect to the different sub-algorithms of the method. For the Remove and the Add Mode the complexity for defining the search space is $O(|N_{out}|)$ and

Table 2: Transposed Threshold vector

6	7	9	10	13	17	15	12	16
-0.013	-0.013	-0.002	0.003	0.005	0.008	0.008	0.011	0.016

Table 3: Contribution Matrix after Threshold Subtraction

	6	7	9	10	13	17	15	12	16
(2,)	0.002	0.002	0.002	-0.028	-0.029	-0.033	-0.033	-0.036	-0.041
(5,)	0.011	0.011	0.0	0.0	0.0	0.0	0.0	0.0	0.0
(11,)	0.012	0.012	0.002	0.021	0.019	-0.008	-0.008	0.013	-0.016
(14,)	0.012	0.012	0.002	-0.003	-0.005	0.016	0.016	-0.011	0.008
(1, 5)	0.0	0.0	0.0	-0.024	-0.024	-0.024	-0.024	-0.024	-0.024
(1, 11)	0.002	0.002	0.002	-0.003	-0.005	-0.033	-0.033	-0.011	-0.041
(1, 14)	0.002	0.002	0.002	-0.028	-0.029	-0.008	-0.008	-0.036	-0.041
(5, 11)	0.011	0.011	0.0	0.024	0.024	0.0	0.0	0.024	0.0
(5, 14)	0.011	0.011	0.0	0.0	0.0	0.024	0.024	0.0	0.024
(11, 14)	0.012	0.012	0.002	0.021	0.019	0.016	0.016	0.013	0.008
(1, 5, 11)	0.0	0.0	0.0	0.0	0.0	-0.024	-0.024	0.0	-0.024
(1, 5, 14)	0.0	0.0	0.0	-0.024	-0.024	0.0	0.0	-0.024	0.0
(1, 11, 14)	0.002	0.002	0.002	-0.003	-0.005	-0.008	-0.008	-0.011	-0.016
(5, 11, 14)	0.011	0.011	0.0	0.024	0.024	0.024	0.024	0.024	0.024

$O(|N_{out}| + |PPR_{WNI}|)$ respectively, with N_{out} being the outgoing neighborhood of the user node and PPR_{WNI} (Algorithm 2, Line 8) the list of potential neighbors from the *ReverseLocalPush* function.

The complexity of the explanation construction is $O(|H|)$ in *Incremental* and $O(2^{|H|} - 1)$ in *Powerset*, where H is the list of selected nodes that was created in the previous step (Add or Remove Mode). The complexity is higher in the *Powerset* algorithm because we are computing the complete powerset of edges in H (at the worst case), while in the *Incremental* mode, we compute *one* potential explanation for each size at every step (at the worst case). However, we avoid exponential costs (the worst case) by considering only solutions that meet the contribution criteria (Equations 6 and 5), checking the candidates in a size ascending fashion. The complexity of the complete solution using the *Incremental* algorithm in Remove and Add mode is $O((|N_{out}|) + |H|)$ and $O(|N_{out}| + |PPR_{WNI}| + |H|)$ respectively. Similarly the complexity using the *Powerset* in Remove and Add mode is $O((|N_{out}|) + (2^{|H|} - 1))$ and $O(|N_{out}| + |PPR_{WNI}| + (2^{|H|} - 1))$ respectively.

The *Exhaustive Comparison* method builds the contribution matrix with a time complexity of $O(2^{|H|} \times (|T| - 1))$. In practice, a similar method as before (threshold application) is applied to avoid the exponential cost. Then, with M being the contribution matrix, all filtered sets of actions are linearly tested. Finally the complexity of the solution in exhaustive Remove mode an exhaustive Add mode is respectively $O(|N_{out}| + (2^{|H|} \times (|T| - 1)))$ and $O((|N_{out}| + |PPR_{WNI}|) + (2^{|H|} \times (|T| - 1)))$.

We note that *EMiGr* depends on the complexity of the Personalised Page Rank computation, and can benefit from optimisation on graph-update computation results in this field of research (see [17] for a very recent study).

The heuristics presented in Algorithms 3, 4, and 5 are correct, i.e., a set of edges returned by one of these algorithm is an explanation. The proof is trivial, by the CHECK test added in the algorithms at lines 18, 25, and 39 respectively.

5.4 Choice of the Method

The goal of both modes (*Add*, *Remove*) is the same: propose a modification on the graph (aka Why-Not Explanation) that would change the output of the recommendation engine, such that the new recommendation be the Why-Not item. However, the space where they are looking is different: In the Remove mode, we search in the user node’s existing neighborhood for a modification. Thus, we want

to use a previous action of the user to explain “Why *WNI* was not recommended?”. On the other hand, in the Add mode, we search for a new, non-existing edge and thus propose to increase the size of the neighborhood of the user node. In this case, we suggest a new action for the user to explain “Why *WNI* was not recommended?”.

The choice of the mode depends on various criteria, the first one being the size of the space to explore. If the neighborhood of the user is too small (low activity user) the algorithm is less likely to find a solution to the problem in remove mode. In this case, the add mode will perform better as it will look for a solution in a wider space. In addition, if the space of the neighborhood of the user is too big (high activity user) the algorithm will consume a lot of time and computation resources. Using the add mode, in this case, may be beneficial. Second, the choice in the mode is linked to the desired output to the end-user and what would be the most meaningful to them. A platform trying to increase user activity may opt for the add mode to suggest new actions to perform. On the other hand, if the user need is to understand the system based on previous actions, then the remove mode may be preferred.

6 EXPERIMENTAL EVALUATION

6.1 Datasets and Setting

Datasets We are using the Amazon Customer Review dataset¹, describing the activity of users on the e-commerce platform. The provided information includes users, items, reviews, ratings (on a scale of 1 to 5), and categories.

We performed some preprocessing on the data set, in order to make it exploitable by our methods and to optimize the experiments. First, we modeled the data set into a directional graph following a similar model as in [11]. Nodes correspond to entities and links correspond to the relationships between them, namely: “rated” (user-item), “reviewed” (user-item), “has-review” (item-review), and “belongs-to” (item-category). The resulting graph contains 11831 nodes and 40552 edges. In Table 4 we provide information on the node degrees, i.e., the number of edges connected to a node, per node type in the resulting graph.

Table 4: Node degree statistics per node type in the graph.

Node Type	# of Nodes	Average Degree	Degree STD
Reviews	2334	2.28	0.7
Categories	32	366.8	291.9
Items	7459	5.4	2.4
Users	120	22.1	2.7

In general, since the graph is directional, the relationship and the direction of those relations need to have a semantic meaning. If a relation can’t be mutual, the edge can’t be bidirectional. An example in Figure 1a shows user interaction as unidirectional. However, if the relation is mutual, the connection can but does not have to be bidirectional. More, since the PPR computation is directional [39], a graph with more possible paths allows less restriction. With respect to our case, we consider any type of relationship to be bidirectional.

Then, we included only good ratings (over 3) to keep only the items appreciated by the user. We also enriched the data set with

¹(Released by Amazon:s3.amazonaws.com/amazon-reviews-pds/readme.html)

review-review links representing the similarity between each pair of reviews and assigning the cosine similarity of the review embedding, generated with Google’s Universal Sentence Encoder [5].

Finally, we randomly sampled 100 users from the set of ‘moderate/active’ users, i.e., those having between 10 and 100 actions in the system. For those users, we extracted their four-hop neighborhood from the original graph, in order to make the computations more feasible and without harming the effectiveness of the recommendations. We call this new data set/graph ‘Amazon Lite’.

Setting We implemented and ran the experiments in Python 3.9, on a machine equipped with Intel Xeon X5670 and 128 go of RAM. The hyper-parameter values were experimentally set as follows: (i) the teleportation probability $\alpha = 0.15$, (ii) the parameter of the Random Walk $\beta = 0.5$, and (iii) the parameter of the reverse/forward local push methods $\epsilon = 2.7e - 8$.

The implemented algorithms are openly available in the following GIT repository: <https://git.cyu.fr/hattolou/why-not-explainable-graph-recommender>

6.2 Experimental Design

We computed the top-10 recommendation list for each one of the 100 users of the Amazon Lite data set. Then, for each user, we computed the Why-Not explanation for each one of the items in his/her recommendation list (except for the first one) as the Why-Not item. Following our privacy-related restrictions detailed in Section 4, explanations can only be part of the subset T_e that includes only user-item edges, both in Remove and Add Mode. Since no different weight is applied to the “rated” (user-item) and “reviewed” (user-item) links, the results in Add Mode can be interpreted indifferently as “the user should have rated A” or as “the user should have reviewed A”. The explanations were computed based on different variants of the algorithms presented in Section 5. More specifically,

- *add_Powerset*, *Powerset* heuristic, Add Mode;
- *add_Incremental*, *Incremental* heuristic, Add Mode;
- *add_ex Exhaustive Comparison* heuristic, Add Mode;
- *remove_Powerset Powerset* heuristic, Remove Mode;
- *remove_Incremental Incremental* heuristic, Remove Mode;
- *remove_ex Exhaustive Comparison* heuristic, Remove Mode.

We also include 2 more methods as comparison baselines:

- **Brute force:** We create a list of all possible edges to remove, sorted by size, and try all of them until a solution is found. This model is able to find the minimal explanation when available because it explores the full space of solution in Remove Mode. Thus, it serves as a comparison point for the explanation size and success rate (two of our metrics described later). Due to the size of the space to explore the process is expected to consume a lot of processing time. In Add Mode the same process could be applied but the size of the solution space is far bigger and the time needed is prohibitive.
- **Exhaustive Comparison direct:** We compute the explanations using the *Exhaustive Comparison* but we terminate the process when the smallest explanation candidate (even if it is not guaranteed to be a correct explanation) is found, thus skipping the “CHECK” process. The goal is to prove the necessity of the “CHECK” step by observing the gap in success

rates between the *Exhaustive Comparison* and the *Exhaustive Comparison* direct versions.

To the best of our knowledge, there are no other related works on the same problem, which explains the absence of competitor algorithms in the comparative evaluation.

To evaluate our work, we analyze the results of the algorithms on three quantifiable metrics:

- **Success rate:** measures the percentage of scenarios, i.e., (user, missing item) pairs, for which an algorithm finds a correct explanation, the best value being 100%. However, as mentioned later in Section 6.4, providing an explanation for a missing item may be infeasible due to the nature of the data.
- **Runtime:** measures the efficiency of an algorithm in seconds. Naturally, the lowest the value, the better the performance.
- **Explanation size:** measures the length (in the number of edges) of an explanation returned by an algorithm. As stated in the literature (e.g., [11]) the shorter the explanation, the better.

6.3 Results

Success Rate. Figure 4 reports the success rates percentages for each different algorithm. Overall, the Add Mode is far more successful than the Remove Mode, and also by comparing the heuristics (remove *Incremental* vs add *Incremental*). The *Exhaustive Comparison* sub-algorithm performs the best with a success rate of 75%.

For the Remove mode, we can first observe a low success rate in general, even in the brute force baseline. Thus, this is a problem linked to the scenario, and not to our algorithms, since no solution exists in most of the cases, because of the limited past user-actions space. Basically, in most of the failure scenarios, the recommended item is a *popular item* (see Section 6.4) and thus just removing user actions cannot lower its popularity in benefit of the Why-Not item, in contrast to the Add mode that allows for creating a stronger network around the Why-Not item.

To allow for a more pertinent experiment for Remove Mode, we present also the success-rate results in the cases when a solution can be found, given the current data structure. These cases can be identified by the success of the brute force algorithm, and the results are reported in Figure 5. In this figure, we can see that the *Exhaustive Comparison* method can perform the closest to brute force, while also the *Powerset* method has a success rate over 90%. Should the use case focus only on the success rate in Remove mode, we can conclude that these sub-algorithms are the most suited.

We also report here the conclusions made w.r.t. the necessity of the check phase in the *Exhaustive Comparison* method. In Figure 5 we can observe a 33% drop of the success rate for the *Exhaustive Comparison* direct version (Section 6.2) of our *Exhaustive Comparison* sub-algorithm. This means that the *CHECK* step in the *Exhaustive Comparison* algorithm is necessary for pruning out false positive explanations, even if it adds up to the runtime of the algorithm, as seen in Table 5.

As a remedy to the low success rate, we could add a post-processing step that informs the user of the reason of failure. This information could be presented as a form of *meta-explanations*, and includes the category of zoma-structure related problem as listed in Section 6.4.

Table 5: Average runtime in seconds per method, (a) in the general case, (b) when an explanation is found, and (c) when no explanation is found. Dark-shaded rows are for Add Mode, light-shaded rows are for Remove Mode and the last two rows are baseline models.

Method	(a)	(b)	(c)
add_Incremental	6.54	8.31	5.78
add_Powerset	57.55	133.96	8.19
add_ex	21618.32	23924.37	14646.56
remove_Incremental	9.07	8.20	9.15
remove_Powerset	287.91	15.32	315.31
remove_ex	173.44	24.48	190.13
remove_ex_direct	25.14	21.81	25.38
remove_brute	908.73	22.37	1008.07

Size. In Figure 6 we present the results on the computed explanations size, which are overall small. Using brute force as a baseline for the Remove Mode, we observe that the *Exhaustive Comparison* method performs the best (lowest size) along with *Powerset*. These results can be compared because of the close success rates, as we report averages. On the other hand, the *Exhaustive Comparison* direct demonstrates relatively better results. However, since it returns a correct explanation for less cases than the rest of the remove mode alternatives, it may have pruned-out longer explanations, affecting its average success rate. In Add Mode, the explanations sizes tend to be close to the smallest possible size (one edge created), except for the *Incremental* heuristic, where the size is bigger (which is also visible in the Remove Mode in the same heuristic).

Time. Table 5 reports the average runtime in seconds per method, in the general case (column a), in the cases when an explanation was found (column b) and in the cases when no explanation was found (column c). In Remove mode, as expected brute force is the longest. The *Exhaustive Comparison* method in Remove Mode is comparable to brute force when successful, but the *Exhaustive Comparison* method is 81.1% faster to respond that no solution can be provided. The *Incremental* heuristic is 46.4% faster than the *Powerset* heuristic, as expected. Note that *Exhaustive Comparison* direct is faster than *Exhaustive Comparison* and this is explained by the early termination condition of this base-line.

In Add mode, the *Exhaustive Comparison* is very time-consuming both when successful and unsuccessful, showing no practical interest when time performance is critical. The *Incremental* heuristic is 93.8% faster than the *Powerset* heuristic. The *Exhaustive Comparison* is taking the most time, as expected, either when an explanation is found or not. If time is an essential criterion this method should be avoided and substituted by the *Incremental* or *Powerset* heuristic.

The general observation is that the computation time is long for most of the methods that are not optimised for runtime, i.e., different than *Incremental*, so when time is a critical parameter, *Incremental* is to be preferred. However, the *Exhaustive Comparison* in Remove mode is a good compromise between runtime and success rate.

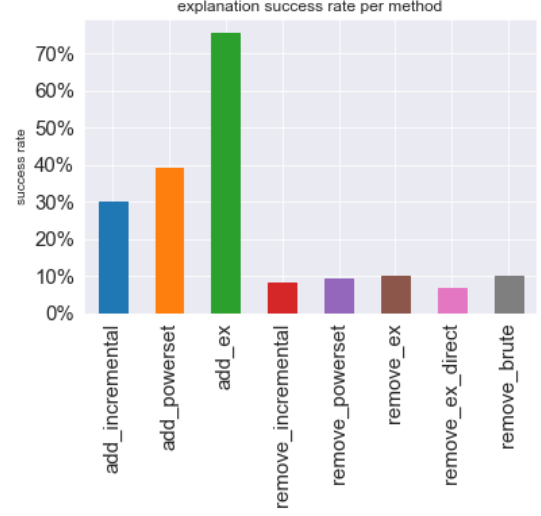


Figure 4: Explanation success rate per method.

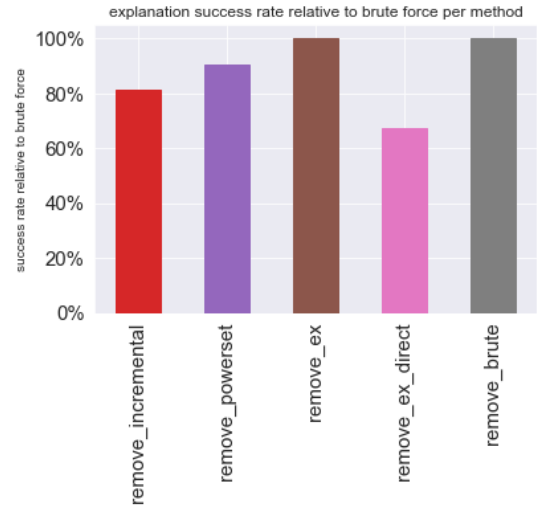


Figure 5: Explanation success rate relative to brute force for remove method.

6.4 Challenges

We now present some challenges in connection with the observed experimental results, especially with respect to runtime and success rate, and give directions for addressing them.

Cold Start And Less Active Users. Our graph-based algorithm takes advantage of the modeling of the activity of users. Yet, if no or little information is present for a user, there will be few edges/candidates for an explanation, leading in the worst case to empty explanations. This is connected to the low success rate that we observe in the experiments.

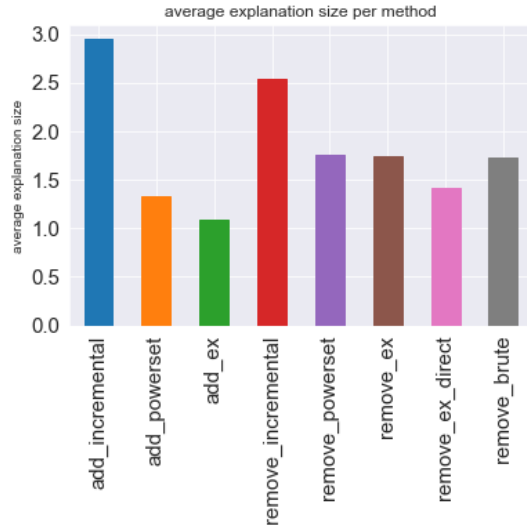


Figure 6: Average explanation size per method.

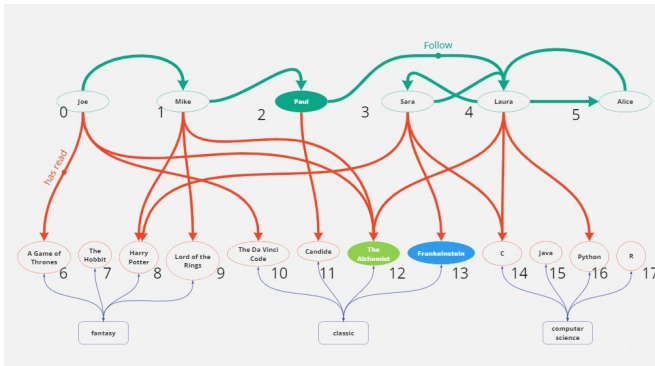


Figure 7: Example of impossible explanation for the Why-Not item (node 13) due to higher popularity of Paul's recommended item (node 12).

Popular Item. In PageRank, by definition, popular items tend to have a high PPR. Thus, popular items are more likely to be recommended to users, who have some previous activity in connection with the category of the popular item. Selecting any other item (than the popular one) as the Why-Not item may generate a case where it is impossible to compute an explanation for it; the Why-Not item will never be more important than the popular one. This is a direct implication of our algorithm design choice to use only the outgoing edges of the user (user actions) as explanations, which may not be enough to influence the popularity of a specific item, which depends also on other users' actions. Nevertheless, we remind that this design choice is deliberately made in order to provide personalised, actionable explanations and to protect other users' privacy. In Figure 7, we illustrate such a use case.

Out Of Scope Item. Another case that may arise is when only additions or only deletions of edges to/from the user node cannot lead to the Why-Not Item being a recommendation. In such cases,

EMiGre could be extended to a combined add/remove mode, which we leave for future work.

Check Step. When searching for an explanation in the Remove mode (similarly for the Add mode), we may be led to removing an edge that lowers the $PPR(u, rec)$ between user u and the recommended item rec , but that also influences in a positive manner not only the position of WNI in the new recommendation list but also that of any other item, that may surpass WNI . So, to avoid false positive explanations we have added a CHECK step in the algorithms that ensures that the explanation is correct, which is time consuming.

7 CONCLUSION

In this work, we proposed *EMiGre*, a graph-based explainable recommendation system tailored for Why-Not explanations, i.e., explaining why a certain item is not recommended for a specific user. *EMiGre* formalises the problem of asking Why-Not questions in this setting and provides actionable explanations in a form of Counterfactual Explanations. More specifically, the explanations constitute a set of either past user-actions or potential new actions that the user should perform in order to place the missing recommendation in the top of the list. We provide different sub-algorithms that respond to different needs; either optimised run-time or explanation size. *EMiGre* is configurable to allow for user-preferences in the type of actions to appear in the explanations, e.g., for addressing privacy concerns or domain constraints. *EMiGre* is implemented over a graph-based RS based on Personalised Page Rank, but can be easily adapted to other user-defined monotonic functions computing the contribution of edges on personalised recommendations. Our analytical experimental study on real data demonstrates the performance of *EMiGre* with respect to a variety of metrics, showcases the desired behaviour of each sub-algorithm, and reveals interesting directions for future work.

Future Work One of the main directions of the future work is to deal with the low success rate. One possible extension would be to mix past and future actions in the explanations, or to not only consider edges but the weight of edges in the graph for the explanations as well. For example, an explanation could be "You should have rated book A with 5 stars to get recommended book B". To assess the acceptance of our explanations by the final users, we aim to conduct a user study replicating a real use case scenario.

REFERENCES

- [1] Krisztian Balog, Filip Radlinski, and Shushan Arakelyan. Transparent, scrutable and explainable user models for personalized recommendation. In *Proceedings of the 42nd international acm sigir conference on research and development in information retrieval*, pages 265–274, 2019.
- [2] Behnoud Abdollahi and Olfa Nasraoui. Using Explainability for Constrained Matrix Factorization. In *Proceedings of the Eleventh ACM Conference on Recommender Systems, RecSys '17*, pages 79–83, New York, NY, USA, August 2017. Association for Computing Machinery.
- [3] Nicole Bidoit, Melanie Herschel, and Katerina Tzompanaki. Query-based why-not provenance with nedexplain. In *Extending database technology (EDBT)*, 2014.
- [4] Nicole Bidoit, Melanie Herschel, and Katerina Tzompanaki. Refining SQL queries based on why-not polynomials. In *8th USENIX Workshop on the Theory and Practice of Provenance, TaPP 2016, Washington, D.C., USA, June 8-9, 2016*, 2016.
- [5] Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St. John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, Yun-Hsuan Sung, Brian Strope, and Ray Kurzweil. Universal sentence encoder, 2018.

- [6] Adriane Chapman and H. V. Jagadish. Why not? In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, SIGMOD '09, page 523–534, New York, NY, USA, 2009. Association for Computing Machinery.
- [7] Zixuan Chen, Panagiotis Manolios, and Mirek Riedewald. Why not yet: Fixing a top-k ranking that is not fair to individuals. *Proc. VLDB Endow.*, 16(9):2377–2390, may 2023.
- [8] Weiyu Cheng, Yanyan Shen, Linpeng Huang, and Yanmin Zhu. Incorporating interpretability into latent factor models via fast influence analysis. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 885–893, 2019.
- [9] Maartje MA De Graaf and Bertram F Malle. How people explain action (and autonomous intelligent systems should too). In *2017 AAAI Fall Symposium Series*, 2017.
- [10] Tim Donkers, Benedikt Loepp, and Jürgen Ziegler. Sequential user-based recurrent neural network recommendations. In *Proceedings of the eleventh ACM conference on recommender systems*, pages 152–160, 2017.
- [11] Azin Ghazimatin, Oana Balalau, Rishiraj Saha Roy, and Gerhard Weikum. PRINCE: Provider-side Interpretability with Counterfactual Explanations in Recommender Systems. *Proceedings of the 13th International Conference on Web Search and Data Mining*, pages 196–204, January 2020. arXiv: 1911.08378.
- [12] Jonathan L. Herlocker, Joseph A. Konstan, and John Riedl. Explaining collaborative filtering recommendations. In *Proceedings of the 2000 ACM conference on Computer supported cooperative work*, CSCW '00, pages 241–250, New York, NY, USA, December 2000. Association for Computing Machinery.
- [13] Glen Jeh and Jennifer Widom. Scaling personalized web search. In *Proceedings of the 12th international conference on World Wide Web*, pages 271–279, 2003.
- [14] Yehuda Koren. Factorization meets the neighborhood: A multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '08, page 426–434, New York, NY, USA, 2008. Association for Computing Machinery.
- [15] Daniel D Lee and H Sebastian Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401(6755):788–791, 1999.
- [16] Yanhong Li, Wang Zhang, Changyin Luo, Xiaokun Du, and Jianjun Li. Answering why-not questions on top-k augmented spatial keyword queries. *Knowledge-Based Systems*, 223:107047, 2021.
- [17] Zihao Li, Dongqi Fu, and Jingrui He. Everything evolves in personalized pagerank. In *Proceedings of the ACM Web Conference 2023*, pages 3342–3352, 2023.
- [18] Peter Lipton. Contrastive explanation. *Royal Institute of Philosophy Supplements*, 27:247–266, 1990.
- [19] Pasquale Lops, Marco de Gemmis, and Giovanni Semeraro. *Content-based Recommender Systems: State of the Art and Trends*, pages 73–105. Springer US, Boston, MA, 2011.
- [20] Scott M. Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 4768–4777, Red Hook, NY, USA, 2017. Curran Associates Inc.
- [21] Julian McAuley and Jure Leskovec. Hidden factors and hidden topics: understanding rating dimensions with review text. In *Proceedings of the 7th ACM conference on Recommender systems*, pages 165–172, 2013.
- [22] Tim Miller. Explanation in artificial intelligence: Insights from the social sciences, 2018.
- [23] Tim Miller. Contrastive explanation: A structural-model approach. *The Knowledge Engineering Review*, 36:e14, 2021.
- [24] Athanasios N. Nikolakopoulos and George Karypis. RecWalk: Nearly Uncoupled Random Walks for Top-N Recommendation. February 2019.
- [25] Xia Ning and George Karypis. Slim: Sparse linear methods for top-n recommender systems. In *2011 IEEE 11th International Conference on Data Mining*, pages 497–506, 2011.
- [26] Dimitris Palyvos-Giannas, Katerina Tzompanaki, Marina Papatriantafyllou, and Vincenzo Gulisano. Erebus: Explaining the outputs of data streaming queries. *Proc. VLDB Endow.*, 16(2):230–242, 2022.
- [27] Gustavo Padilha Polleti and Fabio Gagliardi Cozman. Explaining content-based recommendations with topic models. In *2019 8th Brazilian Conference on Intelligent Systems (BRACIS)*, pages 800–805. IEEE, 2019.
- [28] Thomas Rebele, Katerina Tzompanaki, and Fabian M. Suchanek. Adding missing words to regular expressions. In Dinh Q. Phung, Vincent S. Tseng, Geoffrey I. Webb, Bao Ho, Mohadeseh Ganji, and Lida Rashidi, editors, *Advances in Knowledge Discovery and Data Mining - 22nd Pacific-Asia Conference, PAKDD 2018, Melbourne, VIC, Australia, June 3-6, 2018, Proceedings, Part II*, volume 10938 of *Lecture Notes in Computer Science*, pages 67–79. Springer, 2018.
- [29] J. Ben Schafer, Joseph Konstan, and John Riedl. Recommender systems in e-commerce. In *Proceedings of the 1st ACM conference on Electronic commerce*, EC '99, pages 158–166, New York, NY, USA, November 1999. Association for Computing Machinery.
- [30] Sungyong Seo, Jing Huang, Hao Yang, and Yan Liu. Interpretable convolutional neural networks with dual local and global attention for review rating prediction. In *Proceedings of the eleventh ACM conference on recommender systems*, pages 297–305, 2017.
- [31] Kostas Stefanidis, Nafiseh Shabib, Kjetil Nørvåg, and John Krogstie. Contextual recommendations for groups. In *Advances in Conceptual Modeling - ER 2012 Workshops*, pages 89–97, 2012.
- [32] Maria Stratigi, Aikaterini Tzompanaki, and Kostas Stefanidis. Why-Not Questions & Explanations for Collaborative Filtering. In *WISE*, Amsterdam, Netherlands, October 2020.
- [33] Zhiqiang Tao, Sheng Li, Zhaowen Wang, Chen Fang, Longqi Yang, Handong Zhao, and Yun Fu. Log2intent: Towards interpretable user modeling via recurrent semantics memory unit. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1055–1063, 2019.
- [34] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. Neural graph collaborative filtering. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, jul 2019.
- [35] Yao Wu and Martin Ester. Flame: A probabilistic model combining aspect based opinion mining and collaborative filtering. In *Proceedings of the eighth ACM international conference on web search and data mining*, pages 199–208, 2015.
- [36] Minghua Xu and Shenghao Liu. Semantic-enhanced and context-aware hybrid collaborative filtering for event recommendation in event-based social networks. *IEEE Access*, 7:17493–17502, 2019.
- [37] Siyu Yao, Jun Liu, Meng Wang, Bifan Wei, and Xuelu Chen. Anna: Answering why-not questions for sparql. In *ISWC (Posters & Demos)*, 2015.
- [38] Hongyang Zhang, Peter Lofgren, and Ashish Goel. Approximate Personalized PageRank on Dynamic Graphs. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1315–1324, San Francisco California USA, August 2016. ACM.
- [39] Hongyang Zhang, Peter Lofgren, and Ashish Goel. Approximate personalized pagerank on dynamic graphs. In *Proceedings of the 22nd ACM SIGKDD International Conference on knowledge discovery and data mining*, pages 1315–1324, 2016.
- [40] Yongfeng Zhang and Xu Chen. Explainable Recommendation: A Survey and New Perspectives. *INR*, 14(1):1–101, March 2020. Number: 1 Publisher: Now Publishers, Inc.