



OLLSCOIL NA GAILLIMHÉ
UNIVERSITY OF GALWAY

Master of Engineering Science Degree By Research (M.Eng.Sc)

An Effective Graph-based Diffusion Method for Top-N Recommendation

Yifei Zhou

January 21, 2025

External Examiner
Dr. Ioana Hulpus

Supervisor
Dr. Conor Hayes

Internal Examiner
Dr. John McCrae



Insight Centre for Data Analytics
School of Computer Science, University of Galway

ABSTRACT

The primary interest of this thesis is to design a graph-based recommendation approach to improve the recommendation result. The traditional recommendation approaches did not address data sparsity and insufficient information utilization issues. Consequently, we are motivated to build a user-item combination graph and apply the graph traversals on that graph to address these problems. Firstly, we use probabilistic graph traversals to solve the data sparsity problem by exploring the indirect relationships between users-to-users and items-to-items. Besides, we investigate graph kernels to effectively measure the similarity between a pair of graph nodes and combine the diffusion kernel with the graph-based recommendation approach. Then, to solve the insufficient information utilization problem, we aim to explore the item's semantic information from knowledge graphs using deterministic graph traversals. We build the semantic inter-item graph and combine it with the graph-based recommendation approach to improve the recommendation result further. Finally, we experiment with our proposed methods on publicly well-known datasets to investigate the recommendation performance.

CONTENTS

Abstract	i
1 Introduction	1
1.1 Motivation	1
1.2 Research Questions & Contribution	2
1.2.1 Research Questions	2
1.2.2 Contribution	3
1.2.3 Structure of Thesis	4
2 Background: Introduction to Recommender Systems	5
2.1 Personalized Recommendation	5
2.2 Content-based Filtering	5
2.3 Collaborative-based Filtering	7
2.3.1 Similarity-based Method	9
2.3.2 Model-based Method	11
2.3.3 Neural Collaborative Method	14
2.3.4 Regression-based Method	14
2.4 Hybrid Filtering	16
2.5 Conclusion	17
3 Background: Graph Models and Algorithms	19
3.1 Graph Basis	19
3.2 Item Model	22
3.3 Graph Traversals	22
3.3.1 Deterministic Graph Traversal	23
3.3.2 Probabilistic Graph Traversal	23
3.4 Conclusion	28
4 Background: Graph Diffusion Kernels	29
4.1 Diffusion Kernels	29
4.2 Neumann series-based Kernels	30
4.2.1 Personalized PageRank Kernel	31
4.2.2 Regularized Laplacian Kernel	32
4.3 Exponential-based Kernel	34
4.3.1 Communicability Diffusion Kernel	34
4.4 Kernels Demonstration	35
4.4.1 Personalized PageRank Kernel	37
4.4.2 Regularized Laplacian Kernel	41
4.4.3 Communicability Kernel	43
4.5 Conclusion	47
5 Methodology: Generating Recommendations Using Graph Diffusion Kernels	48
5.1 Model Construction	48
5.2 Model Instantiation	54
5.2.1 RecWalk* + PPR	54

5.2.2	RecWalk* + LAP	59
5.2.3	RecWalk* + DR	59
5.3	Conclusion	60
6	Evaluation of Graph Diffusion Kernel-based Recommendations	61
6.1	Data	61
6.2	Algorithms	62
6.2.1	Non-item Baseline Algorithms	62
6.2.2	Item-based Baseline Algorithms	63
6.2.3	Kernel-based Algorithms	63
6.2.4	Conclusion	63
6.3	Experiments	63
6.3.1	Evaluation	64
6.3.2	Results	64
6.4	Conclusion	68
7	Extending Kernel-based Recommendations with a Knowledge Graph	70
7.1	Motivation	70
7.2	Knowledge Graphs	70
7.3	Data	72
7.3.1	Standard Recommendation Datasets & Results	72
7.3.2	Entity Linking	72
7.4	Knowledge Graph Extraction	73
7.4.1	DFSPARQL	74
7.5	Knowledge Graph Filtering	77
7.5.1	Shared Categorical Labels	78
7.5.2	Item Embeddings	79
7.5.3	Embedded-item Model	79
7.6	KGRecWalk* Model	81
7.7	Conclusion	86
8	Conclusion & Future Work	87
8.1	Conclusion	87
8.1.1	Discussion	87
8.1.2	Thesis Conclusion	88
8.1.3	Contribution	89
8.2	Drawbacks	89
8.3	Future Work	90
	Appendices	91
A	Unfiltered Knowledge Graph	92
A.1	Unfiltered ML-300K knowledge subgraph	92
A.2	Unfiltered FB-MS knowledge subgraph	93
A.3	Unfiltered LT knowledge subgraph	95
B	Filtered knowledge subgraph	97
B.1	Filtered ML-300K knowledge subgraph	97
B.2	Filtered FB-MS knowledge subgraph	102
B.3	Filtered LT knowledge subgraph	107

DECLARATION

I declare that this thesis, titled "*An Effective Graph-based Diffusion Method for Top-N Recommendation*", is composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification.

Galway, January 21, 2025

Yifei Zhou

1

INTRODUCTION

With the rapid development of information technology, recommender systems have been increasingly applied in many digital platforms from different domains like e-commerce and on-line entertainment over the last few decades. Such a recommender system helps users filter the negative information and find relevant items as personalized recommendations. Meanwhile, semantic data enriches the meaning and contexts of data, making it more understandable and usable. In this thesis, we will explore the methods that combine the semantic information of items into the recommender system to improve the recommendation accuracy.

1.1 MOTIVATION

Recommender systems aim to suggest appealing items (e.g., songs, movies, books, and games) to a user by analysing his prior preferences. Cognitively speaking, recommender systems are such things that ‘If you like that, you may also like this’, as shown in Figure 1.1.

As the demand for recommender systems has increased dramatically during the last few decades, improving recommendation performance has become a crucial problem. The recommendation algorithms are the core part of a recommender system. A recommender system uses recommendation algorithms to recommend items to the user. Ranking and rating prediction [BA11] are the two fundamental recommendation strategies for measuring the accuracy of a recommender system. The rating prediction strategy aims to give an explicit rating (e.g., 1 to 5 stars) that a user would like to choose. In contrast, the ranking prediction strategy aims to generate a list of items ordered by their predicted likelihood that a user will interact with and concentrates on the position of the relative items within the ranked list. In this thesis, we only use the ranking



Figure 1.1: Recommender Systems [Fan21]

prediction strategy.

The traditional recommender systems (e.g., collaborative filtering) only focus on the user-item ratings or interaction data to build the user-item interaction matrix. However, data sparsity is a significant challenge in designing a recommender system, as most elements are missing in the matrix. Insufficient information usage is another issue. For example, memory-based methods only use the user-item relationships but do not consider the deep inter-user and inter-item relationships. Additionally, traditional recommender systems ignore semantic information about users and items, which can prevent an understandable and usable interpretation. Data sparsity and insufficient information usage are two significant factors limiting the recommendation's usability.

This thesis proposes a graph-based recommendation approach to address these three problems. A graph is a data structure that models the pairwise relationships between objects. A graph projects the users and items onto nodes and their interactions onto links. We can build a user-item bipartite or inter-item graph or combine these two as a user-item combination graph. Then, we can run a probabilistic graph traversal like a random walk on the user-item combination graph, starting with a user node, and the user can choose to stop at one item node unrated by the user as the recommendation after a few steps of moves. Afterwards, we plan to use graph kernels to measure the similarity of item nodes on the combination graph to find the items that mostly fit the user's preference. At the end, we plan to explore the semantic information about items from the knowledge graph. We use the item's semantic information to measure the semantic relatedness or similarity between a pair of items and improve the recommendation results.

1.2 RESEARCH QUESTIONS & CONTRIBUTION

1.2.1 Research Questions

This thesis aims to build a graph-based recommendation engine that incorporates the items' semantic information into the standard recommendation data and addresses these problems existing in the traditional recommender systems to improve recommendation accuracy. Therefore, our main research question is: **Can a graph-based recommendation engine combined with the items' semantic information perform better than using the standard recommendation data?** Several essential aspects need to be addressed during our research process. Therefore, we divide our main research question into three sub-questions and solve them step-by-step in the following chapters.

Research Question 1: Which diffusion kernel performs best on the standard recommendation graph?

The first stage of our work is to build a graph-based recommendation engine. We first create a user-item bipartite graph and an inter-item graph from the user-item interaction matrix. Then, we combine these two graphs as a user-item combination graph. Afterwards, we apply the probabilistic graph traversals starting with user nodes and terminating at the item nodes. As graph

kernels measure the similarity or relatedness between a pair of graph nodes, we explore different diffusion kernels and investigate which diffusion kernel outperforms the recommendation result.

Output: *Graph-based Diffusion Method for Top-n Recommendation*

Research Question 2: Can we extract a sub-knowledge graph using the most relevant relations from the source knowledge graph in order to enhance the recommendation data?

The second stage of our work is divided into two parts. The first part involves extracting the sub-knowledge graph for each item of the standard recommendation dataset. We start by finding the item's entity URI using the standard entity linking technique. Then, we extract and filter the local sub-knowledge graph for each item based on the set of candidate relations that reveal the item's semantic properties.

Research Question 3: Can the embedding representation be constructed from the knowledge graph to accurately and precisely enhance the semantic representation of the items?

The next part transforms the sub-knowledge graph into items' embedding representation. We use the knowledge graph embedding technique to translate the sub-knowledge graph onto items' embedding vectors. Then, we can calculate the semantic similarity or relatedness scores for a pair of items using the items' embedding vectors and build an embedded-item graph as the embedded-item model instead of using the collaborative filtering user-item matrix.

1.2.2 Contribution

Overall, our contributions are developing a graph-based recommendation engine and exploring items' semantic information from knowledge graphs to improve the recommendation accuracy consisting of:

1. Review some of the traditional collaborative filtering algorithms, like neighbour- and model-based algorithms. Reproduce the results of some traditional recommendation algorithms on datasets and compare them with others' works.
2. Design a graph-based recommender engine (RecWalk*) that builds a user-item combination graph. Apply the probabilistic graph traversals (like random walks) starting with the user nodes and terminating at the item nodes on the graph to generate personalized recommendation results.
3. Investigate graph kernels (like diffusion kernels) as an efficient tool for measuring the similarity or relatedness of graph nodes and integrate them into the graph-based recommender engine.
4. Extract the items' semantic information from the knowledge graph (e.g., DBpedia) using deterministic graph traversals (like DFS) and convert them onto semantic embedding vectors using

knowledge graph embedding to accurately measure the semantic relatedness for a pair of items.

1.2.3 Structure of Thesis

The structure of the thesis is as follows. The following chapters discuss related works and describe preliminary materials from recommender systems, knowledge graphs, and data augmentation that form the fundamentals of our work.

Chapter 2 presents a systematic overview of recommender systems, with a particular focus on the personalized recommendation strategy and various types of recommendation algorithms. This Chapter underlines the limitations of traditional recommender systems and the compelling rationale for adopting the graph-based method.

Chapter 3 describes the graph models and graph algorithms. We introduce the concepts of graphs and give two graph representations of the standard recommendation dataset: a user-item bipartite graph and an inter-item graph. We also give two types of graph traversals, including deterministic and probabilistic, which are used to solve graph-based problems.

Chapter 4 discusses diffusion kernels, which accurately and effectively measure a pair of graph nodes' similarity or relatedness. To illustrate how a diffusion kernel works, we apply the probabilistic graph traversals to the inter-item model.

Chapter 5 introduces our methodology. We propose a graph-based random walk framework and demonstrate how this framework integrates with a diffusion kernel as a kernel-based recommendation method.

Chapter 6 sets up our experimental procedures in detail. We first select some classical item-based and non-item-based recommendation algorithms as the baseline recommendation algorithms. Then, we introduce some well-known standard recommendation datasets and conduct experiments on these datasets to compare and analyze their results.

Chapter 7 mainly discusses items' semantic information extraction and embedding transformation from knowledge graphs. We start by stating the entity linking technique, which connects the items of the standard recommendation dataset to the entity nodes of the knowledge graph. Then, we introduce the knowledge graph extraction, filtering, and embedding techniques to obtain the concise knowledge subgraph for items. At the end of this chapter, we extend the standard random-walk framework, RecWalk*, (as discussed in Chapter 5), using the knowledge subgraphs.

Chapter 8 summarizes all contributions discussed in this thesis and answers all research questions, as we introduced in this chapter. We also give a systematic evaluation of the method proposed in this thesis, including its advantages and disadvantages. At the end of this chapter, we state our plans for future work.

2

BACKGROUND: INTRODUCTION TO RECOMMENDER SYSTEMS

Nowadays, many people have chosen online shopping and entertainment as their primary ways to spend their spare time. Meanwhile, most people are unlikely to make decisions effectively and wisely when they face massive information – *Information Overload*. This term was argued in the research of modern information management from Bertram Gross’s book in 1965 [Gro65]. Some researchers developed *recommender systems* (RSs) to provide personalized recommendations to satisfy the specific preferences of different users. This chapter begins with the personalized recommender systems and introduces some classical algorithms. Then, we move on to the graphs (in Chapters 3 and 4), introducing the concepts of graph data structures, graph models, graph kernels, and graph algorithms and providing an example to illustrate each concept. Finally, we briefly discuss the *knowledge graphs* (KGs).

2.1 PERSONALIZED RECOMMENDATION

Recommender Systems are techniques for information filtering and searching [KAU16], aiming to find interesting items and filter out negative items for users. There are two major types of recommender systems: *Non-personalized Recommendation* and *Personalized Recommendation* [HAJ+22]. As a global recommendation strategy, the non-personalized recommendation also refers to the popularity-based or average-based recommendation, suitable for new users or users with fewer interaction records in the database. For instance, a system returns the top-10 selling books to all users. Meanwhile, the personalized recommendation strategy can recommend items to a user based on his or her preferences. Such a method explores the interactions between users and items. Traditionally, a personalized recommender system includes three different types of methods [WCL+23]: *content-based filtering* (CB), *collaborative filtering* (CF), and *hybrid methods*.

2.2 CONTENT-BASED FILTERING

Content-based filtering is the oldest recommendation strategy, in that an item is represented as a vector of content features, and a user can be represented using a weighted vector of the same content [PBo7]. The item description is a vector of contents. The user description is also a vector of contents which has the same dimension representing the features as item description. Given a user, we compare the similarity between the user’s feature vector and all items’ feature vectors using a similarity measure metric (such as the cosine measure). We then return the top items with the highest similarity scores as the recommended list. Principally, all users are independent

of each other in the content-based recommendation method. Figure 2.1 below demonstrates an example of a content-based recommendation. In this example, we provide three apps with six categories: Education, Casual, Health, Timewaster, Science R Us, and Healthcare, as shown in the first three rows, and a user profile with preferred item categories. Each cell in the app's profile represents whether an app belongs to a specific class. For example, the first app (the first row on the table) occupies the 'Educational' and 'Science R Us' two cells. Each cell represents the user's preferences for this user profile, such as Education, Science R Us and Healthcare products.

Table 2.1 (a) below shows the app profile, where each cell gives the association score [0-1] of an app belonging to a category. Table 2.1 (b) shows the user profile, where the one entries represent the user's preferences; otherwise, they are zeros.

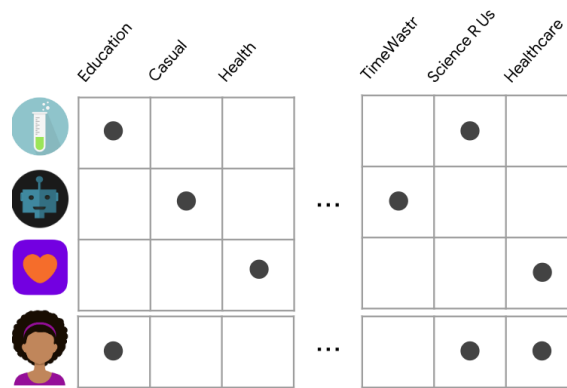






Figure 2.1: Example of content-based recommendation [Goo24]

	[0.75	0	0	0	0.51	0]
	[0	0.55	0	0.76	0	0]
	[0	0	0.84	0	0	1]

(a) Items feature vectors

	[0.6	0	0	0	0.25	0.33]
---	------	---	---	---	------	-------

(b) Users feature vector

Table 2.1: Items feature vectors and user feature vector

Next, we calculate the similarities between the user feature vector and all items' feature vectors using the cosine similarity measure [SKK+01], which returns a similarity score from the interval $[-1,1]$. In general, due to the non-negativity of each feature, the final similarity score is from the interval $[0,1]$.

Table 2.2 shows the Cosine similarity results of three items' feature vectors compared with the user feature vector ranked by their similarity scores in descending order. Obviously, the item with the score (0.873) will be returned to the user as the result of content-based recommendation.




	0.873
	0.377
	0.094

Table 2.2: Cosine similarity results

The advantages and disadvantages of the content-based recommendation are clear. As far as the benefits are concerned, the recommendation process for each user is independent. Therefore, there is no need to learn the data about other users. In addition, the recommendation results are varied with different users' preferences. The feature vectors can be changed with the chosen user's or items' profiles. However, the drawbacks are obvious to see. One drawback is that such an approach can only recommend items focused on content that the user has 'liked before'; in this situation, it cannot recommend App-2 because the user has not previously rated other apps with the 'Casual' or 'TimeWaster' category. Another drawback is that we cannot provide recommendations unless we obtain good descriptions of items.

2.3 COLLABORATIVE-BASED FILTERING

Collaborative filtering (CF) is one of the most popular recommendation strategies. This method collects ratings data from all users and puts the users with similar preferences into the same groups [ERK+11]. The prediction score of an item to a user is based on the ratings of other users with similar interests, and the ratings they gave to that item. One hypothesis under this method is that if a person M has the same interests as another person N on an item, M is more likely to have N's opinion on a different item than that of a randomly selected person [RRS10]. In other words, similar people like similar items. The significant difference between content-based filtering and collaborative filtering appears in user engagement, as illustrated in Figure 2.2 below.

Collaborative filtering uses data from all users and items, but the content-based filtering uses the data from the users and the data from the items, separately. For the collaborative-filtering method, a user-item rated matrix is constructed from the database. Figure 2.3 below illustrates an example of the user-item rated matrix. Within this figure, the rows represent three users (John, Tom, and Conor), and the columns represent nine items (food). Each entry within this table represents the preference or rating of the item a user gives. Each rating is on the scale from 1-5 which is called the 'Likert Scale' [Lik32]. The question marks in the cells represent the unknown data (missing values or the predicted ratings).

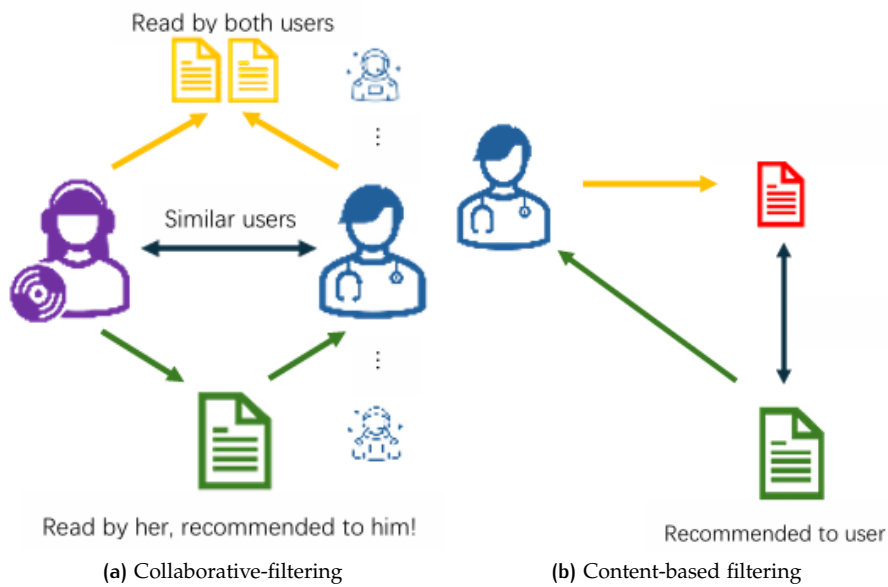


Figure 2.2: The difference between content-based Filtering and collaborative Filtering













									
John 	5	1	3	?	5	2	?	4	5
Tom 	?	?	?	3	1	?	?	1	?
Conor 	4	?	5	?	5	?	?	?	3

Figure 2.3: User-item rated matrix

There are two major types of user feedback: Explicit and Implicit feedback [JSK10]. Explicit feedback, accurately and explicitly represent a (good or bad) preference score that a user has given to an item (e.g., a user has rated a favourite song as 5 out of 5 which means the user has a very good preference on this song; a user only has rated a movie as 1 out of 5 which means the user probably dislike this movie);

Collaborative Filtering can be divided into two approaches: similarity-based and model-based approaches.

Similarity Metric	Explanation
Cosine	$\cos(\mathbf{a}, \mathbf{b}) = \frac{\mathbf{a} \cdot \mathbf{b}}{\ \mathbf{a}\ \cdot \ \mathbf{b}\ }$
Pearson Correlation Coefficient	$PCC(\mathbf{a}, \mathbf{b}) = \cos(\mathbf{a} - \bar{\mathbf{a}}, \mathbf{b} - \bar{\mathbf{b}})$
Euclidean Distance	$\sum_{i=1}^n \sqrt{(\mathbf{a}_i - \mathbf{b}_i)^2}$

Table 2.3: Similarity metrics in memory-based filtering

2.3.1 Similarity-based Method

The similarity-based method, called memory-based or neighbour-based, compares and calculates the similarity between users and items [SKK+01]. The representation of users and items is the row-based or column-based vectors of the user-item matrix. The standard vector similarity measures include cosine [SKK+01], pearson correlation coefficient (PCC) [CKT10], and euclidean distance [CLS90]. Table 2.3 below explains the similarity metrics. These three similarity metrics all produce a symmetric similarity matrix. We can calculate the similarities of row-based vectors to build a user-to-user similarity matrix as user-based CF or calculate the similarities of column-based vectors to construct an item-to-item similarity matrix as item-based CF.




John 	[5	1	3	0	5	2	0	4	5]
Tom 	[0	0	0	3	1	0	0	1	0]
Conor 	[4	0	5	0	5	0	0	0	3]

Figure 2.4: User row-based vectors

Figure 2.4 shows the row-based vectors for three users. The non-zero represents the rating of the item the user gave, and all zero entries correspond to the question marks as the unknown ratings in Figure 2.3. Table 2.4 shows the user similarity matrix using the cosine measure.

User	John	Tom	Conor
John	-	0.265	0.845
Tom	0.265	-	0.174
Conor	0.845	0.174	-

Table 2.4: Cosine similarity matrix

In this example, the self-similarity to the user itself is omitted. John and Conor are more similar in this metric than any other pair of users. Sometimes, we set a threshold to filter out low-scored links between unrelated users. Some metrics (e.g., Pearson) may produce a negative similarity score for a pair of unrelated users, and those links will not be considered.

$$r_{u,m} = \frac{\sum_{i \in u(k)} \text{sim}(u, i) * r_{i,m}}{\sum_{i \in u(k)} |\text{sim}(u, i)|} \quad (2.3.1)$$

Eq 2.3.1 [SKK+01] above shows a ‘weighted average approach’ to compute the predicted rating score ($r_{u,m}$) for an unseen item (m) to the user (u) as the user-based collaborative filtering. The predicted score is calculated by the weighted sum of the user similarity $\text{sim}(u, i)$ and the rating to the item ($r_{i,m}$) scaled by the sum of user similarities for all top- k most similar users. In this equation, the (k) is the number of selected similar users to the target user (u). The user-based collaborative filtering is called the UserKNN (or ItemKNN [SKK+01] for item-based collaborative filtering). Figure 2.5 below illustrates the user-based collaborative recommendation process to the user ‘Conor’. In this example, we set the number of the nearest neighbours as 1. Eqs 2.3.2 - 2.3.5 calculate the prediction score for each unseen item to the user ‘Conor’. Based on the cosine similarity result in Table 2.4, ‘John’ was most similar to ‘Conor’. The prediction scores of all unseen items for Conor were computed using John’s ratings.

$$r_{\text{Conor},2} = \text{sim}(\text{Conor}, \text{John}) \times r_{\text{John},2} = 0.845 \times 1 = 0.845 \approx 0.85 \quad (2.3.2)$$

$$r_{\text{Conor},4} = \text{sim}(\text{Conor}, \text{John}) \times r_{\text{John},4} = 0.845 \times 0 = 0 \quad (2.3.3)$$

$$r_{\text{Conor},6} = \text{sim}(\text{Conor}, \text{John}) \times r_{\text{John},6} = 0.845 \times 2 = 1.69 \quad (2.3.4)$$

$$r_{\text{Conor},7} = \text{sim}(\text{Conor}, \text{John}) \times r_{\text{John},7} = 0.845 \times 0 = 0 \quad (2.3.5)$$

$$r_{\text{Conor},9} = \text{sim}(\text{Conor}, \text{John}) \times r_{\text{John},8} = 0.845 \times 4 = 3.38 \quad (2.3.6)$$

In Figure 2.5, the bold-font values are the prediction scores of all unseen items to the user ‘Conor’, and the normal-font values are the scores for all rated items to the user. The item (marked as red) was the optimal recommended item returned to the user with the highest prediction score (3.8). We chose this algorithm as one of the baseline recommendation algorithms introduced in











									
Conor 	4	0.85	5	0	5	1.69	0	3.38	3

Figure 2.5: User-based recommendation result to the user Conor

Chapter 5.

Implicit feedback only records whether the user has interacted with the items (e.g. has listened to a song). The degree of user preference for items needs to be clarified in implicit feedback. However, implicit feedback can be used to estimate a user's preference score. For instance, a high preference score could be attached to a song that is listened to frequently. Due to the limited explicit information of a dataset, many collaborative-filtering approaches are based on implicit feedback data.

2.3.2 Model-based Method

However, data sparsity is a crucial problem in traditional memory-based collaborative approaches. For instance, most entries within the user-vector or item-vector are zeros. Therefore, the vector similarity comparison seems as time-consuming and space-consuming as the vectors with high dimensionality (e.g., a vector with 100,000 elements). The model-based methods address this problem and improve data processing efficiency and recommendation accuracy.

In this thesis, we demonstrate matrix factorization as one typical application of the model-based methods. This method decomposes a large and high-dimensional sparse matrix into several dense and low-dimensional matrices [KBV09]. The significant advantage is that we can efficiently compute the similarities between users and items with low-dimensional matrices. For instance, Pure singular value decomposition (PureSVD [CKT10]) decomposes a real or complex matrix into three low-dimensional matrices, including two low-dimensional orthonormal matrices and a diagonal matrix. Eqs 2.3.7 and 2.3.8 below defines PureSVD:

$$\text{PureSVD}(\mathbf{R}) = \mathbf{U} * \mathbf{\Lambda} * \mathbf{Q}^T, \mathbf{P} = \mathbf{U} * \mathbf{\Lambda} \quad (2.3.7)$$

$$\text{PureSVD}(\mathbf{R}) = \mathbf{P} * \mathbf{Q}^T \quad (2.3.8)$$

Within Eq 2.3.7, the user-item matrix (\mathbf{R}) is decomposed into three sub-matrices, \mathbf{U} (an orthogonal matrix), $\mathbf{\Lambda}$ (a diagonal matrix), and \mathbf{Q} (an orthogonal matrix). The matrix (\mathbf{P}), the product result of \mathbf{U} and $\mathbf{\Lambda}$, represents the user-latent features, and the matrix (\mathbf{Q}) represents the item-latent features. Therefore, the user-item matrix (\mathbf{R}) could be represented as the dot product of \mathbf{P} and \mathbf{Q} . Figure 2.5 below illustrates the SVD decomposition process of the user-item matrix in

Figure 2.4. In this example, we set the number of factors as two.

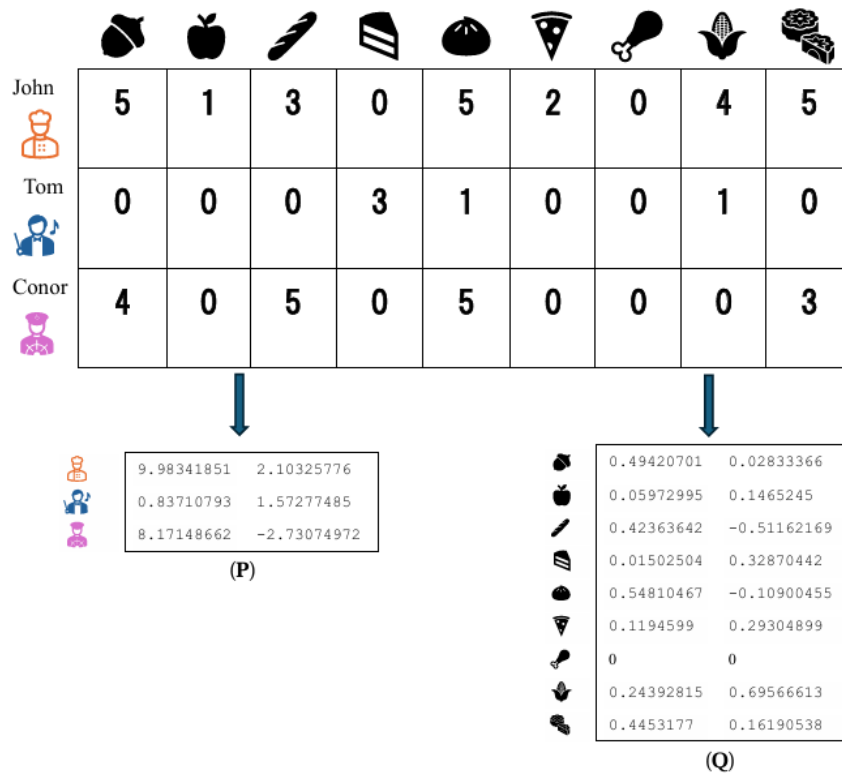


Figure 2.6: User-item matrix factorization (P: User-latent vectors and Q: Item-latent vectors)

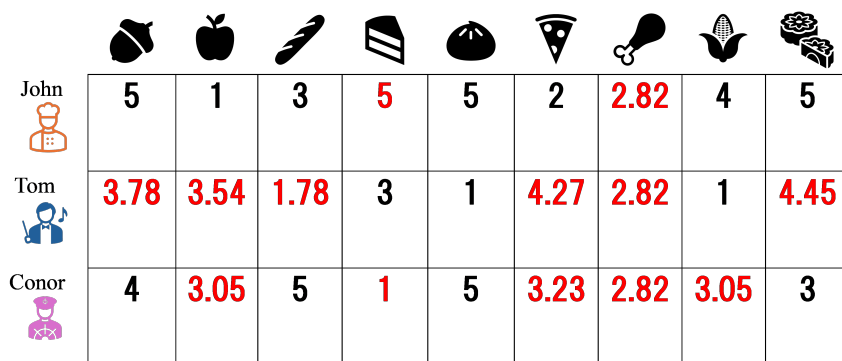


Figure 2.7: PureSVD recommendation illustration (Ratings rescaled into 1-5)

Hence, like the vector similarity measure used in memory-based recommendation methods, the dot product between a user-latent vector and an item-latent vector can be considered the similarity score between the user and the item. In other words, this dot product result reflects the preference of an item given by a user. Figure 2.7 illustrates the scaled predicted ratings from 1 to 5. The black entries are the users' ratings, and the red entries are the predicted ratings. PureSVD provides a straightforward way to decompose a sparse user-item matrix onto a dense user latent

matrix and a dense item-latent matrix so that we can easily calculate the dot product of the vectors as the prediction score. Therefore, we adopted PureSVD as one of our model-based baseline algorithms, which will be introduced in Chapter 5.

2.3.3 Neural Collaborative Method

The earlier versions of the collaborative filtering recommender systems were lazy learning methods, which did not create explicit models from data, like neighbour-based methods. Gradually, model-based methods, called eager-based methods, assemble the data into models to make predictions. Recently, with the rapid development of deep learning, the neural collaborative filtering (NCF) framework [HLZ+17], has been applied to solve a recommendation task in collaborative filtering. This framework, as shown in Figure 2.8 below, has shown significant performance on implicit data. Two well-known algorithms are multiple-layer perceptron (MLP) and generalized matrix factorization (GMF). MLP is a multiple-layer, fully-dense neural network (DNN), and the GMF is a single-layer neural network integrated with the user-latent and item-latent features as the embedding layer weights. The inputs for both algorithms are the user's and item's one-hot representation, respectively, and the output for both algorithms is a probability indicating whether a user liked an item.

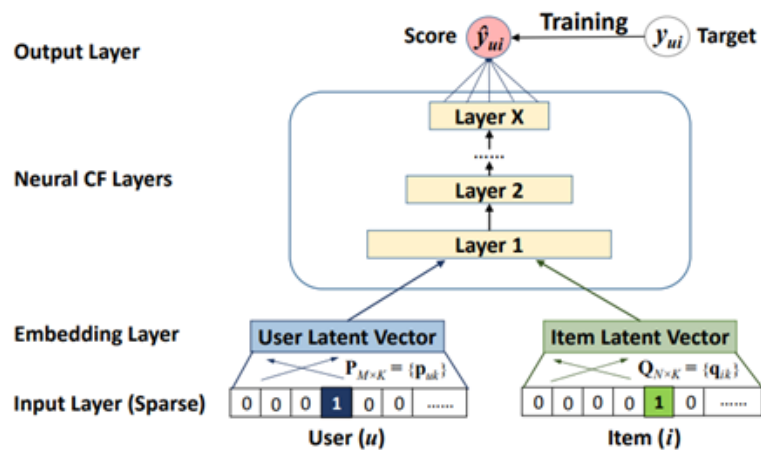


Figure 2.8: Neural collaborative filtering framework [HLZ+17]

The NCF framework converts a collaborative filtering problem into a classification task. This framework consists of four components: an input layer accepting one interaction from a user and an item; an embedding layer creating user latent and item latent vectors; a few neural CF layers for fitting the neural networks; and an output layer that outputs the prediction score a user gives to an item. We first train the neural networks with all user-item interactions from the database and then test with the unknown interactions between users and items. This thesis will use MLP and GMF as two baseline NCF algorithms to compare with other algorithms, as discussed in Chapter 5.

2.3.4 Regression-based Method

Lastly, model-based collaborative filtering can be performed using a regression-based method. Ning and Karypis [NK11] argued with a method, the sparse linear method (SLIM), aiming to solve an inter-item matrix (which will be introduced in Chapter 3) from the explicit or implicit

user-item interaction matrix using the Elastic-net regression. Eq 2.3.9 below illustrates the elastic net regression framework.

$$\begin{aligned} &\text{minimize} && \frac{1}{2} \|\mathbf{R} - \mathbf{R}\mathbf{W}\|_{\mathbb{F}}^2 + \frac{\beta}{2} \|\mathbf{W}\|_{\mathbb{F}}^2 + \lambda \|\mathbf{W}\|_{\mathbb{F}}^1, \\ &\text{subject to} && \mathbf{W} \geq 0, \text{ and } \text{diag}(\mathbf{W}) = 0 \end{aligned} \tag{2.3.9}$$

As shown in Eq 2.3.9, \mathbf{R} represents the explicit or implicit user-item matrix, and \mathbf{W} is the inter-item matrix representing the similarity between any pair of items. The elastic-net regression framework consists of three components, the loss function ($\|\mathbf{R} - \mathbf{R}\mathbf{W}\|_{\mathbb{F}}^2$), the L2-normalization of the inter-item matrix ($\|\mathbf{W}\|_{\mathbb{F}}^2$), and the L1-normalization of the inter-item matrix ($\|\mathbf{W}\|_{\mathbb{F}}^1$). β and λ are the coefficients of the L2-normalization and L1-normalization, respectively. They aimed to minimize the loss to figure out an optimal non-negative inter-item matrix, and the diagonal elements of the inter-item matrix were reset to zeros, ignoring the influence of the items on themselves.

2.4 HYBRID FILTERING

Content-based filtering considers the single user only but requires enough descriptive information about users or items. Collaborative filtering considers the user-item interactions only, with no engagement with the content information about users or items. Hybrid recommender systems fuse content-based and collaborative filtering features [JPL04]. Such a method not only mines the interactions between users and items but also takes advantage of the content information of users or items to improve the recommendation accuracies.



Figure 2.9: Hybrid-filtering framework

Figure 2.9 illustrates the hybrid-filtering framework. There are two similar users (blue and purple) in this example. The left-half side of this diagram represents collaborative filtering, where two yellow items on the top of the diagram are the items co-rated by both users. The right-half side of the diagram shows content-based filtering where the red item at the right-up corner of the diagram is the item rated by the blue user. The green item at the bottom of the diagram is recommended to the blue user via content-based filtering and to the purple user via collaborative filtering, respectively.

However, the major challenge of hybrid filtering is obtaining external content data about users or items. This thesis addresses this problem by using knowledge graphs as introduced in Chapter 6. A knowledge graph, as a topological knowledge base, is considered a good resource for enhancing the description of users or items [FŞA+20]. Each graph node has a real meaning, such as a person, a book, or an abstract concept, and each edge represents a type of relation connecting two entities like friendship or partner. The generic knowledge graphs include DBpedia [ABK+07], YAGO [RSH+16], and WordNet [Fel05]. Figure 2.10 shows a snapshot of DBpedia. As shown in this Figure, the knowledge graph is a directed graph where entities are nodes (such as 'MONA LISA', 'JAMES', and 'PARIS') and links are the relations between entities (such as 'is a', 'is a friend of', and 'has lived in').

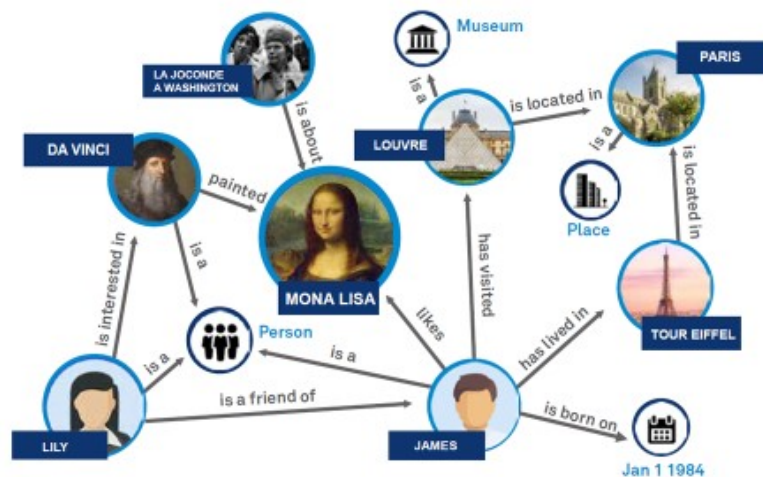


Figure 2.10: Knowledge graph visualization example (DBpedia) [Cha21]

For a standard collaborative filtering task, the items are mapped to entity nodes based on items' descriptive information (e.g., name) in a knowledge graph via a type of *entity linking technique* [HRN+13]. Further, we can extract items' semantic information from the knowledge graph and then convert them into continuous vector spaces via *Knowledge Graph Embedding (KGE)* [WMW+17] as will be discussed in Chapter 6. This technique not only simplifies the manipulation but also preserves and extends the topological structure of the knowledge graph. Afterwards, we can fuse the item's embedding and user-item collaborative information as a hybrid recommender system.

In recent years, many outstanding works have applied semantic knowledge graphs to recommender systems to improve overall recommendation accuracy. In 2016, Tommaso et al. [NOT+16] presented SPrank, a novel hybrid recommendation algorithm, to compute the top-N recommendation effectively. They first extracted a knowledge subgraph from DBpedia via the standard entity linking technique based on semantically correlated relationships. And then, they combined the user-item bipartite graph and the knowledge subgraph as a combination graph. They explored all possible paths between users and items by executing several random walks on the combination graph. Afterwards, they created a feature vector for each user-item interaction instance in the standard recommendation dataset, where each feature represents the occurrence number of a path. Ultimately, they applied all feature vectors to the Random Forest model to keep the best paths by training different decision trees parallelly. Based on that idea, we are motivated to design a graph model that merges the user-item interaction information from the standard recommendation graph and the semantic information from the knowledge graph.

2.5 CONCLUSION

Overall, we first introduced the concept of recommender systems, which are the two standard personalized recommendation algorithms, including content-based and collaborative filtering. The collaborative filtering algorithms are divided into memory-based and model-based algo-

rithms. Further, we demonstrated some applications based on a real user-item rating database and revealed their connections and differences. Next, we introduced neural-based approaches like the NCF framework. Afterwards, we moved onto the regression-based approaches like SLIM. Eventually, we covered the hybrid recommendation strategy that incorporates content-based and collaborative filtering benefits to improve the accuracy of the recommendation. A knowledge graph may be a good resource for extending the semantic descriptions of items in the databases.

Admittedly, traditional content-based and collaborative methods have improved recommendation accuracy in recent decades. However, these methods did not consider the deep relations between users-to-users or items-to-items implied in the database. Therefore, we propose a graph-based method to address this problem. The next chapter will introduce some basic concepts about graphs.

3

BACKGROUND: GRAPH MODELS AND ALGORITHMS

We have discussed the traditional personalized recommendation algorithms in Chapter 2, from content-based and collaborative filtering to neural-based and hybrid methods. However, these methods did not explore the external relationships between users and items. In this thesis, we are going to explore how graphs allow us to model the relationships between users and the relationships between items. We will demonstrate the techniques for making recommendations based on these graph data structures. A graph-based method can conveniently make recommendations. This chapter will first concentrate on the concepts of graphs, then move on to the graph models to represent user-item data, and finally, introduce the graph algorithms in the recommendation task.

3.1 GRAPH BASIS

A graph represents relationships between entities, among users and users, items and items, or users and items. Topologically, a graph is a data structure consisting of nodes and edges, where nodes are entities such as people, events, places or objects, and edges are the relations between a pair of nodes [Gou13]. This thesis first declares some notations and symbols to introduce the concept of a graph. For example, a bold-font upper letter represents a matrix (e.g., \mathbf{R}); a normal-font upper letter represents a graph (e.g., G) or a set (e.g., V); and a bold-font lower letter represents a vector (e.g., \mathbf{r}). Eq 3.1.1 below gives the mathematical definition of a graph. As seen in this equation, a graph is denoted as (G) consisting of a set of nodes (V) and edges (E). An adjacency matrix is a square matrix where each entry indicates if an edge exists between a pair of nodes for an unweighted graph and the degree of strength of an edge in a weighted graph.

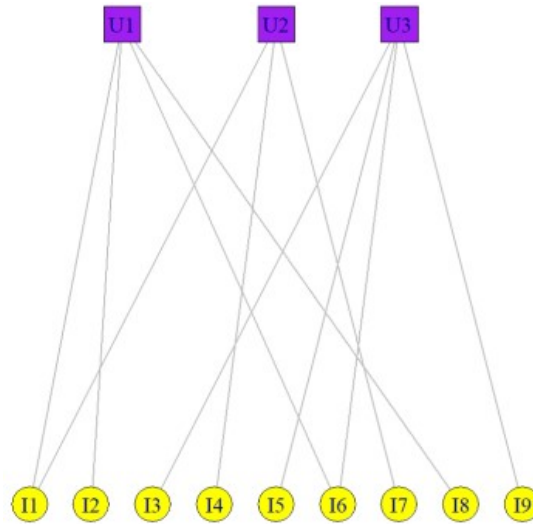
$$G = \langle V, E \rangle, \quad V = \{v | v \in \text{vertices}(G)\}, \quad E = \{(v_1, v_2) | v_1, v_2 \in V\} \quad (3.1.1)$$

For a standard collaborative recommendation task, the user-item matrix (\mathbf{R}) can be represented by a user-item bipartite graph (G) where nodes are the users and items, and links are their interactions, like ratings. Within a bipartite graph, each user node is independent of other user nodes, and each item is independent of other item nodes. In other words, there are no links between users-to-users or items-to-items within a bipartite graph. Figure 3.1 (a) shows the user-item interaction matrix with explicit rating. To simplify the calculation, Figure 3.1 (b) shows

the user-item bipartite graph based on the implicit feedback version of the user-item matrix in Figure 3.1 (a). This user-item bipartite graph contains three user nodes (purple square nodes) from 'U₁' to 'U₃' and nine-item nodes (yellow circle nodes) from 'I₁' to 'I₉'. Figure 3.1 (c) shows the user-item bipartite graph's adjacency matrix (\mathbf{W}). Compared with other types of graphs, the adjacency matrix of a bipartite graph is a *block matrix* [ref] where the adjacency matrix is divided into four blocks from left to right and top to down: the user-user block, the user-item block, the item-user block, and the item-item block, respectively. Eq 3.1.2 shows the block form of the user-item bipartite graph's adjacency matrix (\mathbf{W}).

	I1	I2	I3	I4	I5	I6	I7	I8	I9
U1	5	0	0	3	0	1	0	1	0
U2	0	2	0	1	0	0	2	0	0
U3	0	0	0	0	1	4	0	0	5

(a) User-item ratings matrix



(b) User-item bipartite graph

	U1	U2	U3	I1	I2	I3	I4	I5	I6	I7	I8	I9
U1	0	0	0	5	0	0	3	0	1	0	1	0
U2	0	0	0	0	2	0	1	0	0	2	0	0
U3	0	0	0	0	0	0	0	1	4	0	0	5
I1	5	0	0	0	0	0	0	0	0	0	0	0
I2	0	2	0	0	0	0	0	0	0	0	0	0
I3	0	0	0	0	0	0	0	0	0	0	0	0
I4	3	1	0	0	0	0	0	0	0	0	0	0
I5	0	0	1	0	0	0	0	0	0	0	0	0
I6	1	0	4	0	0	0	0	0	0	0	0	0
I7	0	2	0	0	0	0	0	0	0	0	0	0
I8	1	0	0	0	0	0	0	0	0	0	0	0
I9	0	0	5	0	0	0	0	0	0	0	0	0

(c) Adjacency matrix (\mathbf{W})

Figure 3.1: User-item bipartite graph and adjacency matrix

$$\mathbf{W} = \begin{bmatrix} \mathbf{o} & \mathbf{R} \\ \mathbf{R}^T & \mathbf{o} \end{bmatrix} \quad (3.1.2)$$

Despite the user-item matrix, the user-item bipartite graph clearly presents the interaction behaviours between users and items. The rating prediction in a collaborative filtering task can be solved by *link prediction* in a bipartite graph. Link prediction is a technique that predicts the existence or likelihood of future links within a dynamic network [KSS+20]. This technique predicts most likely missing links from a subset of non-existing edges. The subset of non-existing edges is sorted based on ranking algorithms to reflect the weights, and the high-weighted edges are newly inserted into the original network. Figure 3.2 below shows an example of the prediction process based on the user-item bipartite network provided in Figure 3.1 (b). In this example, both user 'U3' and 'U1' have rated the item ('I6'), and they probably have similar tastes in their preferences. The user 'U1' rated the item 'I8', but the user 'U3' did not. Thus, we predict that there is a link connecting the user 'U3' and the item 'I8' in the graph.

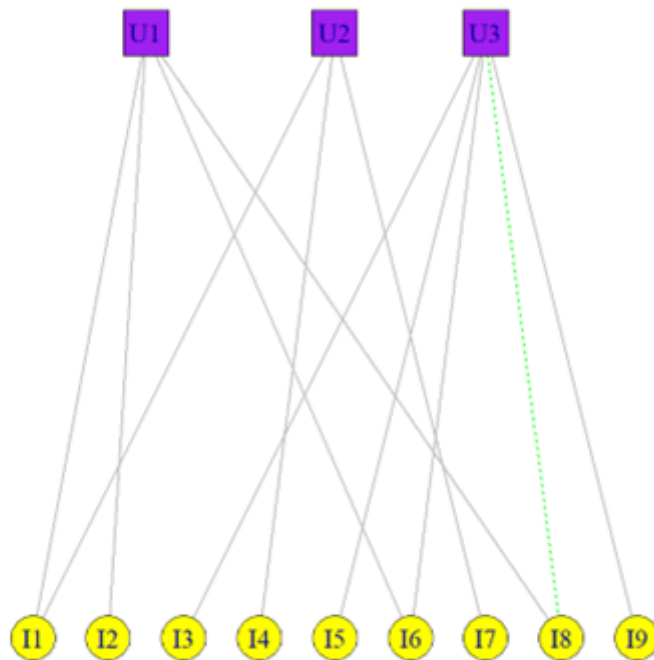


Figure 3.2: Link prediction example

3.2 ITEM MODEL

Although a bipartite graph models the relations between users and items, it is often beneficial in a recommendation task to model the relations between users and users or items and items. For instance, a user has rated an item, and we want to find an item most closely related to the item rated by the user. We can build an item model to simulate this relationships. An item model simulates the transformation from the user-item bipartite graph to the inter-item graph. The transformation calculates the relationships between items that the user-item bipartite graph does not convey. Thus, an inter-item interaction matrix or an inter-item graph can represent an item model. An item model represents the similarity between a pair of items. There are several ways to construct an item model. For instance, an item model can be built from memory-based collaborative filtering methods such as cosine and pearson [CKT10]. Besides, we can also build an inter-item graph by representing items as vectors to reduce the dimensionality of the user-item matrix, as used in PureSVD. Figure 3.3 (a) and (b) show the two representations of an item model as an inter-item matrix and an inter-item graph, respectively, via cosine based on the user-item matrix in Figure 3.1. Due to the symmetry of the cosine similarity measure, the adjacency matrix is symmetrical. Therefore, the inter-item graph is undirected.

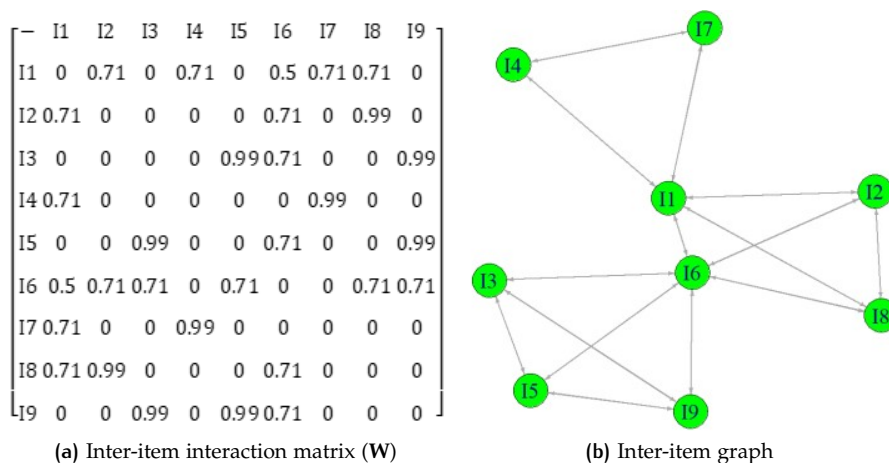


Figure 3.3: Two representations of an item model

Therefore, we have two representations of graphs of recommendation data: a *bipartite graph* represents the user-item raw ratings, and an *inter-item graph*, which is transformed from the user-item bipartite graph, represents the connections and similarities between items and items.

3.3 GRAPH TRAVERSALS

In the last chapter, we examined the neighbour-based and model-based collaborative filtering algorithms. To illustrate how they contribute to a recommendation task, we will introduce graph traversals based on two representations of graphs (user-item bipartite graph and inter-item graph). We will start with the concept of graph traversal.

A *graph traversal* manipulates or instructs a move from one node to another node on a graph [Eve11; Pea05]. Then, we introduce two types of graph traversals [Pea05]: *deterministic* and *probabilistic*. Deterministic traversals involve a known sequence of nodes we are going to visit, such as breadth-first search and depth-first search. On the other hand, probabilistic traversals are those where we are unclear about the sequence of nodes we are going to visit, like random walks. This distinction will help us gain a clear understanding of these traversal types.

3.3.1 Deterministic Graph Traversal

As we mentioned in the introduction part of section 3.3, *depth-first search* (DFS [BW84]) is a type of deterministic graph traversal approach to extract the paths in order starting from a node in a graph. For instance, we will use the DFS approach to extract the paths from a knowledge graph to build a knowledge subgraph for an item, as will be introduced in Chapter 6.

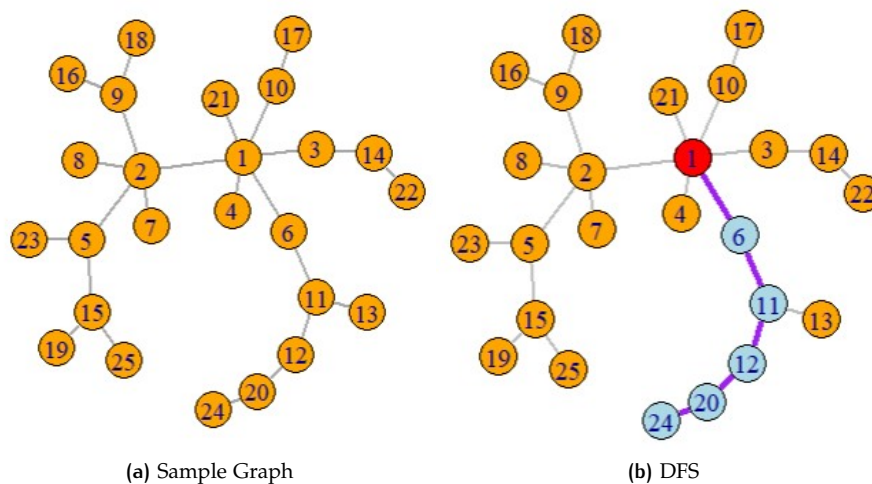


Figure 3.4: DFS traversing example

Figure 3.4 (a) and (b) show an example of a DFS traversing approach based on an undirected graph. In this example, the DFS approach starts with node '1' and explores until it reaches the farthest node '24'.

3.3.2 Probabilistic Graph Traversal

As we discussed in the introduction part of section 3.3, the probabilistic graph traversal explores a sequence of unexpected nodes. A *random walk* [Pea05], argued by Pearson in 1905, illustrates the probabilistic graph traversal. A *walk* [Eve11] consists of a sequence of ordered nodes followed by edges where each node and each edge can be visited repeatedly. For a recommendation task, we aim to produce the user with his unexpected items as the recommendation results. Therefore, we apply the probabilistic graph traversal on the inter-item graph. *Standard PageRank (SPR)*, proposed by Larry Page and Sergey Brin [LM06], is a random walk model that simulates a random web surfer to identify the pages the surfer would likely end up on if they followed the chosen randomly out-coming links. This process is iterated several times. Initially, Google search engines used the SPR algorithm to rank web pages in their search results. The SPR algorithm uses

a probability of teleporting (α) that allows a surfer to jump to a random node rather than following an out-link from the currently occupied node by the surfer. We called the teleportation probability as a damping factor in this thesis. By default, the teleportation probability is set as 0.85.

For a random walk, each move from one node to another node is determined by a transition probability. A *transition probability* gives a likelihood that a random walker moves from one node to another node. In the early 20th century, Andrey Markov presented Markov chain [Sen96], a stochastic probability model to declare the transition probability of each move in a random walk. Each node is viewed as a state within a Markov chain, and each move is considered an event.

Figure 3.5 below provides a visual representation of a random walker based on the Markov chain model. In this example, we use the inter-item graph shown in Figure 3.3 (b). To simplify the process, we omit the weight associated with each edge and only consider the number of in-coming and out-coming links of each node. The probability of transitioning from one node to all other nodes is the reciprocal value of the number of outgoing links from that node. The walker starts at the (red) node 'I5' and ends at the node 'I7' (blue) after three independent moves. The probability of this random walk $P(5 \rightarrow 6 \rightarrow 1 \rightarrow 7)$ is calculated as follows in eq 3.3.1 below.

$$P(5 \rightarrow 6 \rightarrow 1 \rightarrow 7) = P(5 \rightarrow 6) * P(6 \rightarrow 1) * P(1 \rightarrow 7) = 0.33 * 0.17 * 0.2 = 0.011 \quad (3.3.1)$$

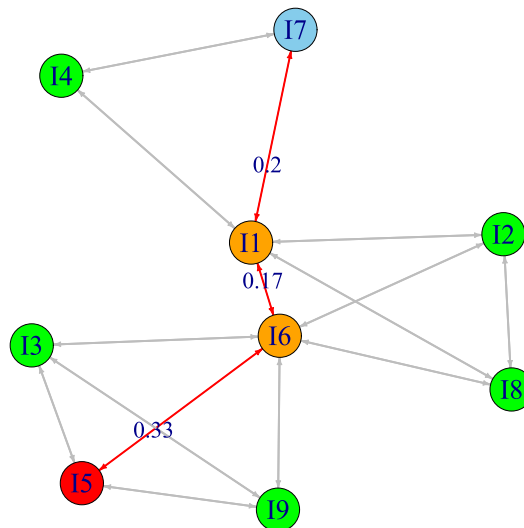


Figure 3.5: Markov chain example

We can use a transition probability matrix to store the transition probabilities among all graph nodes [Sen96]. Eq 3.3.2 below gives the generic definition of the transition probability matrix (P).

The transition probability matrix has the same size as the adjacency matrix of a graph. In this matrix, each row expresses the probability of transitioning from one node to all other nodes.

$$\mathbf{P} = \begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1n} \\ p_{21} & p_{22} & \cdots & p_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ p_{n1} & p_{n2} & \cdots & p_{nn} \end{bmatrix} \quad (3.3.2)$$

Eq 3.3.3 below shows the transition probability matrix based on the inter-item graph in Figure 3.5. For instance, the first row represents the probability of transitioning from node 'I1' to all connected nodes ('I2', 'I4', 'I6', 'I7', and 'I8'), respectively.

$$\mathbf{P} = \begin{bmatrix} & \text{I1} & \text{I2} & \text{I3} & \text{I4} & \text{I5} & \text{I6} & \text{I7} & \text{I8} & \text{I9} \\ \text{I1} & 0 & 1/5 & 0 & 1/5 & 0 & 1/5 & 1/5 & 1/5 & 0 \\ \text{I2} & 1/3 & 0 & 0 & 0 & 0 & 1/3 & 0 & 1/3 & 0 \\ \text{I3} & 0 & 0 & 0 & 0 & 1/3 & 1/3 & 0 & 0 & 1/3 \\ \text{I4} & 1/2 & 0 & 0 & 0 & 0 & 0 & 1/2 & 0 & 0 \\ \text{I5} & 0 & 0 & 1/3 & 0 & 0 & 1/3 & 0 & 0 & 1/3 \\ \text{I6} & 1/6 & 1/6 & 1/6 & 0 & 1/6 & 0 & 0 & 1/6 & 1/6 \\ \text{I7} & 1/2 & 0 & 0 & 1/2 & 0 & 0 & 0 & 0 & 0 \\ \text{I8} & 1/3 & 1/3 & 0 & 0 & 0 & 1/3 & 0 & 0 & 0 \\ \text{I9} & 0 & 0 & 1/3 & 0 & 1/3 & 1/3 & 0 & 0 & 0 \end{bmatrix} \quad (3.3.3)$$

As previously discussed, the standard PageRank algorithm is an iterative process. We can use a location vector $\boldsymbol{\pi}_{(t)}$, a row-wise vector, to record the probability distribution for each node after each iteration of a random surfer. For example, Eq 3.3.4 defines the probability distribution vector where the entry $p(k)_t$ represents the probability that the surfer will be at the node (k) after (t) iterations. $\boldsymbol{\pi}_{(0)}$ represents the initial probability distribution for the location of a random surfer. Eq 3.3.5 gives the iterative update equation of the probability distribution vector for all items in each iteration. In this equation, $\boldsymbol{\pi}_{(t)}$ represents the probability distribution vector after (t) iterations. The probability distribution vector at the iteration (t) is determined by the probability distribution vector at the iteration (t - 1).

(α) is the teleportation probability, as we introduced before, and $\mathbf{1}$ is a row vector filling with ones with the size of the number of nodes (n) in the graph. (\mathbf{P}) is the transition probability matrix of a random surfer traversing from the current node to another connected node following their out-coming link. Therefore, a surfer landing on a node has two options to move. The surfer has the probability of (α) to choose an out-coming link moving from the current node onto another connected node with their transition probability declared in (\mathbf{P}) ; the surfer has the probability

$$\pi_{(3)} = \left[0.26 \quad 0.02 \quad 0.02 \quad 0.05 \quad 0.01 \quad 0.14 \quad 0.05 \quad 0.34 \quad 0.04 \quad 0.01 \quad 0.02 \quad 0.05 \right] \quad (3.3.8)$$

$$\pi_{(5)} = \left[0.28 \quad 0 \quad 0.01 \quad 0.09 \quad 0 \quad 0.18 \quad 0.02 \quad 0.36 \quad 0.02 \quad 0 \quad 0.01 \quad 0.03 \right] \quad (3.3.9)$$

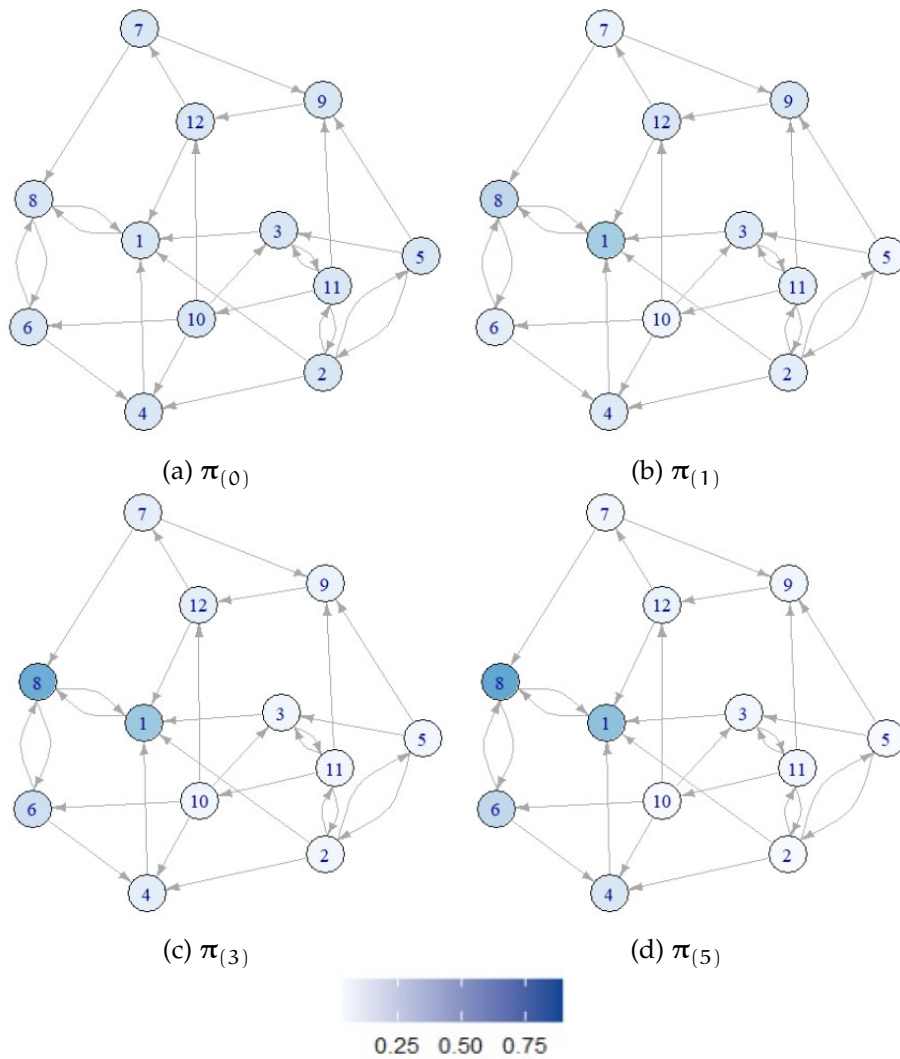


Figure 3.7: SPR: landing probability distribution ($\alpha = 0.85$)

In addition, we also visualized the probability distribution vector after one, three, and five iterations of running the standard PageRank algorithm, as shown in Figure 3.7 below. Figure 3.7 (a) denotes the initial probability distribution vector $\pi_{(0)}$. Figures 3.7 (b)–(f) shows the probability distribution vectors $\pi_{(1)}$, $\pi_{(3)}$, and $\pi_{(5)}$ after one, three, and five iterations of running the standard PageRank algorithm, respectively. In this Figure, a node's colour reflects the node's PageRank score. We concluded that the probability distribution vector converged after five iterations, and the item node '8' has the highest PageRank score. Therefore, the item '8' would be recommended to the user.

3.4 CONCLUSION

Overall, this chapter aims to introduce a straightforward graph-based recommendation strategy that explores the rich information between users-to-users and item-to-items from the primary user-item database, which is not accomplished by the traditional collaborative filtering methods like memory-based or model-based approaches, as discussed in Chapter 2.

We began by introducing the graph basis to understand the graph structures. We then transitioned to the practical application of graph models, including the representations and constructions of the user-item bipartite and inter-item graphs. Next, we introduced two types of graph traversals: deterministic and probabilistic. We gave the DFS algorithm as an illustration of the deterministic graph traversal and the SPR algorithm as an illustration of the probabilistic graph traversal.

Afterwards, we applied the SPR algorithm on the inter-item graph and simulated a random surfer to make non-personalized recommendation result for the user in a recommendation task. By default, the SPR algorithm adopted ($\alpha=0.85$) as the damping parameter. This means that the surfer has the high probability to follow the out-coming links moving onto other connected nodes and the low probability jumping to one of the graph nodes randomly. Notably, such a method delivers the non-personalized recommendation result to the user because each item node has the same initial landing probability as the reciprocal value of the number of nodes within the initial probability distribution vector.

The SPR algorithm provides a non-personalized recommendation strategy because such a method delivers a global ranking of vertices. However, we expect that each user will receive a personalized recommendation result. Thus, we will introduce kernel-based methods in the next chapter. A surfer will start with the user's historically rated item nodes, which are regarded as the seed items. Despite the fact that a surfer has a probability of choosing to move from the current node to another connected node via their outgoing link, we will constrain the surfer restarting to one of the seed nodes rather than a random graph node.

4

BACKGROUND: GRAPH DIFFUSION KERNELS

We introduced two dataset graph representations in the previous chapter: user-item bipartite and inter-item graphs. We also introduced graph algorithms, including the standard PageRank algorithm that delivers a global ranking of vertices as a non-personalized recommendation strategy. Compared with the traditional collaborative-filtering algorithms, we have seen that a graph-based method is an effective recommendation strategy for the user because such a method captures the in-depth relations between items. In this chapter, we will introduce a kernel-based method that can provide personalized recommendation results to a user. Such a method aims to explore the unrated item nodes by the user in the inter-item graph as the recommendation result. Similar to the approach introduced in SPR in Chapter 3, we will use a surfer to simulate a random walk in the inter-item graph. There are two core ideas underlying the kernel-based method: a user can personalize his historically rated items as the seed nodes in the initial landing probability distribution vector, and a surfer can decide to jump back to one of the seed item nodes rather than a random item node after the number of moves. We will start by introducing diffusion kernels and the types of diffusion kernels. Then, demonstrate how a diffusion kernel works with the random walks as a kernel-based method on an inter-item graph to make personalized recommendations to a user based on an actual inter-item graph.

4.1 DIFFUSION KERNELS

As recalled from Chapter 3, the SPR algorithm provides a global ranking of item nodes within an inter-item graph via random walks. However, it fails to deliver personalized recommendations to different users. This is where the understanding of diffusion kernels comes into play. A *kernel*, also known as a *graph kernel*, argued by Kondor and Lafferty [KL02], is a function that computes and compares the similarity, relatedness, or distance between a pair of graph nodes. In these methods, a random surfer starts at a given small subset of graph nodes and moves randomly, following the transition probability as a biased random walk. A diffusion kernel, a graph kernel, operates the energy diffusion process over a net. *Diffusion*, a physical concept, refers to the net movement of a substance (e.g., energy or heat) from an area of high concentration to an area of low concentration [KL02] or the information propagation process on the graph. Importantly, there is a hypothesis that the total energy or information on the graph is constant during the propagation or diffusion process.

In this chapter, we will cover different types of diffusion kernels and provide an example of each kernel based on the actual graph to illustrate how it works with random walks on the inter-

item graph to effectively generate recommendations to the user, starting at the user's rated item nodes. Section 4.2 will introduce two Neumann-series-based kernels: personalized PageRank and laplacian kernel; section 4.3 will discuss an exponential-based diffusion kernel: communication kernel.

As we mentioned previously in the introduction section of this chapter, a diffusion process describes the energy or information propagation process on the graph. During a diffusion process within a graph, a node can act as a role of either transiting information into connected nodes or receiving information from other nodes for each iteration. Therefore, the diffusion result at an iteration can be viewed as the accumulative sum of the diffusion results from all previous iterations. We use a probability distribution vector to record the diffusion scores of all graph nodes for each iteration of the diffusion process. For each diffusion-based approach, we multiply the user's initial landing probability vector (\mathbf{r}) with the diffusion-based results as the final landing probability distribution vector proposed by Karypis and Nikolakopoulos [NK19]. We pick up the top-N unrated items with high landing probabilities as the recommendation per user.

4.2 NEUMANN SERIES-BASED KERNELS

Neumann series-based kernels [SC04] are one of the popular diffusion kernels. Such type of kernels is derived from the Neumann series that sums up the number of repeated applications of an operator. PageRank and laplacian kernels are the Neumann series-based kernels; each can be represented as a Neumann series. In a recommendation task, the kernel-based approaches aim to find a convergent state after executing a diffusion process on the inter-item graph starting with the subset of the item nodes rated by the user. Eq 4.2.1 shows the generic iterative form of the Neumann-series-based diffusion kernels. In this equation, (\mathbf{T}) represents a matrix, (α) indicates a probability that a random walker moves from the current occupied node to another node following the link, and (t) denotes the iteration epoch number of the diffusion process (the number of steps of a random walk). For the SPR algorithm, a random surfer has the probability of (α) moving from the currently occupied item node to another connected item node following the outgoing link and the probability of $(1 - \alpha)$ moving onto a completely random item node; for the Personalized PageRank algorithm, a random surfer has the probability of (α) moving from the currently occupied item node to another connected item node following the out-coming link, and the probability of $(1 - \alpha)$ moving back to one of the seed item nodes. By default, the teleportation probability (α) is set as 0.85, meaning the random surfer will return to one of the seed nodes typically after four or five steps.

As discussed in Chapter 3, we use a landing probability distribution vector to record the landing probability of each node at each iteration in a diffusion process.

Alternatively, the Neumann-series-based kernels can also be transformed into a linear equation, as shown in Eq 4.2.3 below. This transformation, which we will now discuss, is a powerful tool in our understanding of the process. We first left-multiply the term $(\alpha\mathbf{T})$ on both sides of Eq 4.2.1; Eq 4.2.2 shows the result. As the (t) increases to a tremendous value, we can consider that (t) is equal to $(t + 1)$. Therefore, there is only a difference in an identity matrix (\mathbf{I}) between Eq

4.2.1 and Eq 4.2.2. Then, we subtract Eq 4.2.2 and Eq 4.2.1 to obtain a linear equation as in Eq 4.2.3. This linear equation, a more efficient approach, allows us to calculate the convergent state distribution vector by solving the linear equation directly.

$$\mathbf{DF}_{\text{VND}} = \sum_{t=0}^{\infty} \alpha^t \mathbf{T}^t = \alpha^0 \mathbf{T}^0 + \alpha^1 \mathbf{T}^1 + \alpha^2 \mathbf{T}^2 + \dots + \alpha^t \mathbf{T}^t + \dots \quad (4.2.1)$$

$$\alpha \mathbf{T}(\mathbf{DF}_{\text{VND}}) = \alpha^1 \mathbf{T}^1 + \alpha^2 \mathbf{T}^2 + \dots + \alpha^t \mathbf{T}^t + \alpha^{t+1} \mathbf{T}^{t+1} + \dots \quad (4.2.2)$$

$$\mathbf{DF}_{\text{VND}} = [\mathbf{I} - \alpha \mathbf{T}]^{-1} \quad (4.2.3)$$

4.2.1 Personalized PageRank Kernel

As the SPR algorithm discussed in Chapter 3 before, it models a random surfer on an inter-item graph. It ranks all items based on the probability of the surfer reaching a specific item in a random walk. However, such a method delivers a global ranking because this algorithm assumes that the user has the same preference for each item; in other words, each item node has the same initial landing probability that is the same for all users. We run the SPR algorithm on the inter-item graph iteratively until reaching a convergent state. We use a landing probability distribution vector to record the landing probability for each graph node at each iteration. This algorithm returns the items with the highest landing probability to the user. From now on, we introduce the personalized PageRank (PPR) kernel [ACR17]. Such kernel first declares the user's historically positively rated items as the seed items. Users can personalize their starting probability using their seed items as a non-uniform distribution rather than using the same starting probability. Then, a teleportation probability declares a likelihood that a surfer moves following the out-coming links within the inter-item graph or jumps back to a seed node.

The SPR kernel only gives a global view of the graph. Such a kernel does not explore the neighbourhood of the items the user have rated because the random walker searches the whole graph and may returns to a completely random graph node after the number of moves. The PPR kernel explores the local graph around the nodes we have rated because the random walker is constrained to move back to one of the seed nodes after the number of moves. Therefore, the PPR kernel provides a personalized recommendation strategy to the user.

We set up a damping factor as a parameter to allow a user to explore the graph, either locally or globally. The PPR kernel uses a damping factor (α)($0 < \alpha < 1$) as the teleportation probability to indicate whether the walker moves to another connected node following the out-coming link from the currently occupied node with a probability of (α) or randomly jumps to one of the seed nodes with the probability of ($1 - \alpha$).

$$\boldsymbol{\pi}_{(t)} = \alpha \boldsymbol{\pi}_{(t-1)} \mathbf{P} + (1 - \alpha) \mathbf{r} \quad (4.2.4)$$

Eq 4.2.4 declares the generic iterative equation of the personalized PageRank kernel. This equation declares the probability distribution vector for each graph node after (t) iterations of diffusion, (\mathbf{P}) is the transition probability matrix, (\mathbf{r}) is the initial landing probability vector of the user, and (α) is the damping factor indicating the teleportation probability that returns to one of the seed nodes. Thus, a random walker has a probability of (α) to move from an item node to another connected item node with a transition probability declared in (\mathbf{P}) within the inter-item graph or a probability of $(1 - \alpha)$ jumping back to a seed item node with a probability of \mathbf{r} .

Formally, Eq 4.2.5 shows each iteration's update equation of the personalized PageRank diffusion kernel's probability distribution vector $\pi_{(t)}$. In this equation, (\mathbf{r}) is the row-wise vector, which includes the items rated by the user as the seed nodes, (\mathbf{P}) is the transition probability matrix, and (α) is the restarting probability. The (\mathbf{r}) is a row-wise vector representing the user's rated items.

$$\pi_{(t)} = \begin{cases} (1 - \alpha)\mathbf{r}, & t = 0 \\ \alpha\pi_{(t-1)}\mathbf{P} + (1 - \alpha)\mathbf{r}, & t > 0 \end{cases} \quad (4.2.5)$$

Eq 4.2.6 shows the personalized PageRank probability distribution vector $\pi_{(t)}$ as a Neumann series, and Eq 4.2.7 gives the linear solution of the PPR kernel.

$$\mathbf{DK}_{\text{PPR}} = \pi_{(t)} = \mathbf{r} \sum_{k=0}^{\infty} (1 - \alpha)\alpha^k \mathbf{P}^k \quad (4.2.6)$$

$$\mathbf{DK}_{\text{PPR}} = \mathbf{r}[\mathbf{I} - \alpha\mathbf{P}]^{-1} \quad (4.2.7)$$

In short, we have seen that the personalized PageRank kernel helps users make personalized recommendations via a random walker starting with the user's seed item nodes. In the next section, we will introduce the laplacian kernel, a different type of Neumann series-based kernel using different information propagation mechanism.

4.2.2 Regularized Laplacian Kernel

Alternatively, the laplacian matrix [Spi12] is also a matrix representation of a graph. We can use the laplacian matrix to measure the similarity between a pair of graph nodes. *Regularized laplacian kernel* (LAP) (known as the forest kernel [Spi12]) allows the random walker to explore the graph locally or globally to make recommendations. The random surfer can explore the nearby neighbourhoods to the seed nodes as the local exploration result or the broader neighbourhoods to the seed nodes as the global exploration result.

A damping factor (α) determines the possibility of the exploration result to a random surfer. A small value of (α) means that the surfer will likely explore the short-distance neighbourhoods to the seed nodes the user likes on the graph. In contrast, a large value of (α) means that the surfer will likely explore the long-distance neighbourhoods to the seed nodes the user likes on the graph. Eq 4.2.8 defines the laplacian matrix (\mathbf{L}) [Bap10], which is the difference between the degree matrix (\mathbf{D}) and the adjacency matrix (\mathbf{W}) of a graph. A degree matrix is a diagonal matrix that records the vertex degrees of a graph in its diagonal entries.

$$\mathbf{L} = \mathbf{D} - \mathbf{W} \quad (4.2.8)$$

Eq 4.2.9 defines the probability distribution vector $\boldsymbol{\pi}_{(t)}$ of the regularized laplacian diffusion kernel (\mathbf{DK}_{LAP}) to the user. We use the initial probability distribution vector (\mathbf{r}), the same as the one we used in the personalized PageRank kernel. During the diffusion process, the random walker can choose to move from one node to another unvisited node within the inter-item graph with the probability (α) following the negative Laplacian matrix ($-\mathbf{L}$) as the transition probability matrix or restart to one of the seed item nodes rated by the user with the probability ($1 - \alpha$). As a type of Neumann-series-based kernel, Eq 4.2.10 gives the linear solution of the Regularized Laplacian kernel.

$$\mathbf{DK}_{LAP} = \boldsymbol{\pi}_{(t)} = \begin{cases} (1 - \alpha)\mathbf{r}, & t = 0 \\ \alpha\boldsymbol{\pi}_{(t-1)}(-\mathbf{L}) + (1 - \alpha)\mathbf{r}, & t > 0 \end{cases} \quad (4.2.9)$$

$$\mathbf{DK}_{LAP} = \mathbf{r}[\mathbf{I} + \alpha\mathbf{L}]^{-1} \quad (4.2.10)$$

Overall, both personalized PageRank and regularized laplacian kernels are Neumann series-based kernels. The significant difference between these two kernels is the declaration of the transition probability matrix, the information propagation mechanism. Similarly, both kernels allow the random walker to move back to one of the seed item nodes rated by the user during the diffusion process, and their convergent states can be solved by linear equations. The next section will discuss the exponential-based kernel and see how the information propagates differently on the graph.

4.3 EXPONENTIAL-BASED KERNEL

4.3.1 Communicability Diffusion Kernel

Instead of using the Neumann series to represent a diffusion kernel, we can use the exponential function to represent a diffusion kernel. Unlike the PPR and LAP kernels, which allow the random walker to move back to one of the seed nodes with a probability, the exponential-based kernels provides both local and global explorations which not allow the walker to jump back to the seed nodes.

Communicability kernel (DR) [KLo2] is a typical application of the exponential diffusion kernels. Such kernel implements Maclaurin's exponential function [Chu07] and scales the probability distribution vector for each iteration of the diffusion process using the factorials of the number of steps and a damping factor. Compared with the damping factor used in the PPR or LAP diffusion kernel, representing a probability from 0 to 1, the DR kernel uses the damping factor to control the landing probability distribution vector.

Eq 4.3.1 shows the generic Maclaurin's exponential function $\exp(x)$ where $o(x^n)$ refers to the equivalent infinitesimal term. Torres [Tor20] proposed a truncated way to calculate the communicability kernel, in which he cut off the accumulative result after the number of steps (t) of the random walk. He replaced the term (x) in the Eq 4.3.1 as the $(\alpha\mathbf{W})$ where (α) is a damping factor and (\mathbf{W}) is the adjacency matrix of the graph, and defined the communicability diffusion kernel as shown in Eq 4.3.2.

This kernel starts with the items the user has liked in the past to make personalized recommendations to the user. Therefore, we multiply with the user's initial landing probability distribution vector (r). If the number of steps of a walk (t) is small, the surfer would explore the short-distance neighbourhoods to the seed nodes as a local exploration; otherwise, the surfer would explore the long-distance neighbourhoods to the seed nodes as a global exploration. However, with the increasing value of the number of steps (t), the accumulative landing probability for each node would become extremely large because we calculate the power of the transition probability matrix. Thus, we will use a damping factor (α) and the factorials of the step number ($t!$) to rescale the landing probability distribution vectors.

As shown in Eq 4.3.2, $\pi_{(t)}$ defines the probability distribution vector of the communicability diffusion kernel \mathbf{DK}_{DR} , where (\mathbf{W}) represents the adjacency matrix of the inter-item graph as the transition probability matrix. The probability distribution vector of each iteration of the diffusion process is normalized by the scaling factor (α) and the factorial of the iteration number ($t!$).

$$\exp(x) = e^x = 1 + x + \frac{x^2}{2!} + \dots + \frac{x^n}{n!} + o(x^n) \quad (4.3.1)$$

$$\mathbf{DK}_{\text{DR}} = \boldsymbol{\pi}_{(t)} = \mathbf{r} \sum_{t=0}^{\infty} \frac{\alpha^t \mathbf{W}^t}{t!} (\alpha > 0) \quad (4.3.2)$$

In short, these three kernels including PPR, LAP, and DR, provide local and global exploration of the graph. They all aim to find the short-distance or long-distance neighbourhoods to the seed nodes. Especially, PPR and LAP allow the surfer to jump back to one of the seed nodes with a probability, while DR not.

We have introduced two different types of diffusion kernels: Neumann series-based kernels and exponential-based kernels. The following section will demonstrate different diffusion kernels on the actual inter-item graph. On the one hand, we will learn that the kernels exhibit different diffusion features on the graph regarding their information propagation process. On the other hand, we will see how a kernel-based approach contributes to the user's personalized result in a recommendation task.

4.4 KERNELS DEMONSTRATION

Karypis and Nikolakopoulos [NK19] only introduced PPR in their work, but we also introduced other two diffusion kernels (LAP in section 4.2 and DR in section 4.3). In addition, our contributions are that we provide a dynamic example for each diffusion kernel based on a real inter-item graph. In this section, we will demonstrate different diffusion kernels on the inter-item graph, starting with the initial probability distribution vector for a user, and see how it works for a recommendation task.

We adopt the directed inter-item graph introduced in Figure 3.6 as the example used in this section. Figure 3.6 assumes that each link has the same weight. As we discussed earlier, the inter-item graph models the similarity of a pair of graph nodes. To simplify the calculation, we assign each link with a random value from 0-1 as their similarity for a pair of graph nodes. Thus, Figure 4.1 below shows the weighted adjacency matrix of the inter-item graph.

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.72 & 0 & 0 & 0 & 0 \\ 0.53 & 0 & 0 & 0.42 & 0.52 & 0 & 0 & 0 & 0 & 0 & 0.75 & 0 \\ 0.48 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.4 & 0 \\ 0.62 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.28 & 0.8 & 0 & 0 & 0 & 0 & 0 & 0.92 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.79 & 0 & 0 & 0 & 0.69 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.7 & 0.24 & 0 & 0 & 0 \\ 0.22 & 0 & 0 & 0 & 0 & 0.92 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.34 \\ 0 & 0 & 0.15 & 0.5 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0.66 \\ 0 & 0.86 & 0.19 & 0 & 0 & 0 & 0 & 0 & 0.16 & 0.12 & 0 & 0 \\ 0.7 & 0 & 0 & 0 & 0 & 0 & 0.72 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Figure 4.1: Weighted inter-item adjacency matrix (\mathbf{W})

In an actual recommendation task, we aim to find the unseen items related to the items the user positively rated in the past. As we said earlier, we use the user's rated items as the seed items to create the initial probability distribution vector. For example, a user has two rated items, '5' and '7'. Thus, graph nodes '5' and '7' will be chosen as the seed items nodes of the user. We assume that the sum of the probability distribution vector is up to one for each iteration in the diffusion process. Since the user has two rated items, the entries corresponding to the nodes '5' and '7' within the initial probability distribution vector would be 0.5, as the $\pi_{(0)}$ shown in Eq 4.4.1. A random walker starts with the initial probability distribution vector and updates the probability distribution vector with the diffusion kernel iteratively. After each iteration, the probability distribution vector is normalized using the L1-normalization $\|\pi_{(t)}\|_1$ to ensure that the sum of the probability distribution vector is up to one.

$$\pi_{(0)} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0.5 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (4.4.1)$$

Sections 4.4.1-4.4.3 show how a random walk works with the diffusion kernel on the actual inter-item graph to produce personalized recommendation results for the user. We use the three kernels, PPR, LAP, and DR, as we outlined before. For each diffusion kernel, a random walker starts with the same initial probability distribution vector.

4.4.1 Personalized PageRank Kernel

We first demonstrated the diffusion process of the personalized PageRank kernel on the graph. As a type of Neumann-series-based kernel, we used the Eq 4.2.6 to decide the optimal damping factor (α) from a set of values 0.1, 0.3, 0.5, 0.7, 0.9 based on two iterations of the PPR diffusion process, as the results are shown in Figure 4.2 below. For each subgraph in Figure 4.2, the colour represents the landing probability of each item node. As we explained earlier, a damping factor is a probability that refers to the likelihood that a random walker moves from the current occupied item node onto another connected item node following the transition probability defined in the inter-item model. Provided the probability is close to 1, the walker will probably move from the current occupied item node onto a connected item node randomly following a transition probability defined in the inter-item model; otherwise, the walker will randomly jump back to one of the seed item nodes.

Interestingly, the information propagated faster and faster on the net as the (α) increased from 0.1 to 0.9, illustrated by the colour change of the two seed item nodes ('5' and '7'). Therefore, the large damping factor (α) indicates that a random walker will likely move to different unvisited item nodes. Therefore, we selected 0.9 as the optimal damping factor for the Personalized PageRank kernel.

Then, we executed the personalized PageRank diffusion process on the inter-item graph with the random walks starting with the initial probability distribution vector. Figure 4.3 (a) - (e) demonstrates the initial probability distribution of the graph, the diffusion probability distribution of the graph after 1, 2, 4, and 12 iterations, and the diffusion probability distribution of the graph solved by the linear equation, respectively. Eqs 4.4.2 - 4.4.6 quantitatively illustrates the landing probability distribution vectors for the graph for each iteration of the diffusion process.

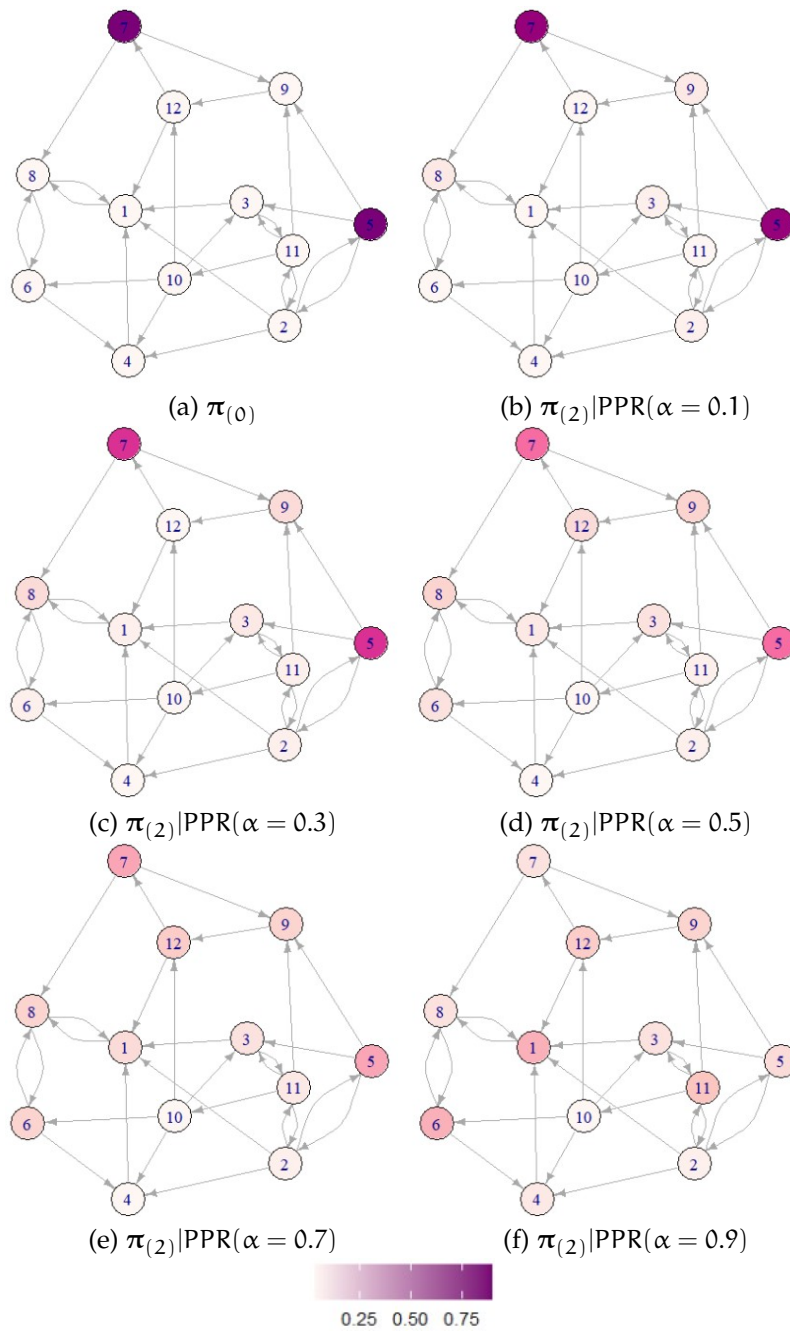
$$\pi_{(1)}^{\text{PPR}(\alpha=0.9)} = \left[0 \quad 0.03 \quad 0.09 \quad 0 \quad 0.26 \quad 0 \quad 0.26 \quad 0.18 \quad 0.17 \quad 0 \quad 0 \quad 0 \right] \quad (4.4.2)$$

$$\pi_{(2)}^{\text{PPR}(\alpha=0.9)} = \left[0.09 \quad 0.02 \quad 0.05 \quad 0.01 \quad 0.15 \quad 0.13 \quad 0.15 \quad 0.1 \quad 0.99 \quad 0 \quad 0.05 \quad 0.16 \right] \quad (4.4.3)$$

$$\pi_{(4)}^{\text{PPR}(\alpha=0.9)} = \left[0.18 \quad 0.03 \quad 0.02 \quad 0.05 \quad 0.06 \quad 0.16 \quad 0.1 \quad 0.25 \quad 0.06 \quad 0 \quad 0.03 \quad 0.07 \right] \quad (4.4.4)$$

$$\pi_{(12)}^{\text{PPR}(\alpha=0.9)} = \left[0.22 \quad 0 \quad 0 \quad 0.14 \quad 0 \quad 0.28 \quad 0.01 \quad 0.33 \quad 0.01 \quad 0 \quad 0 \quad 0.01 \right] \quad (4.4.5)$$

$$\pi_{(1)}^{\text{PPR}(\alpha=0.9)} = \left[0.18 \quad 0.03 \quad 0.02 \quad 0.17 \quad 0.03 \quad 0.22 \quad 0.02 \quad 0.26 \quad 0.03 \quad 0 \quad 0.03 \quad 0.01 \right] \quad (4.4.6)$$

Figure 4.2: PPR: damping factor (α) investigation

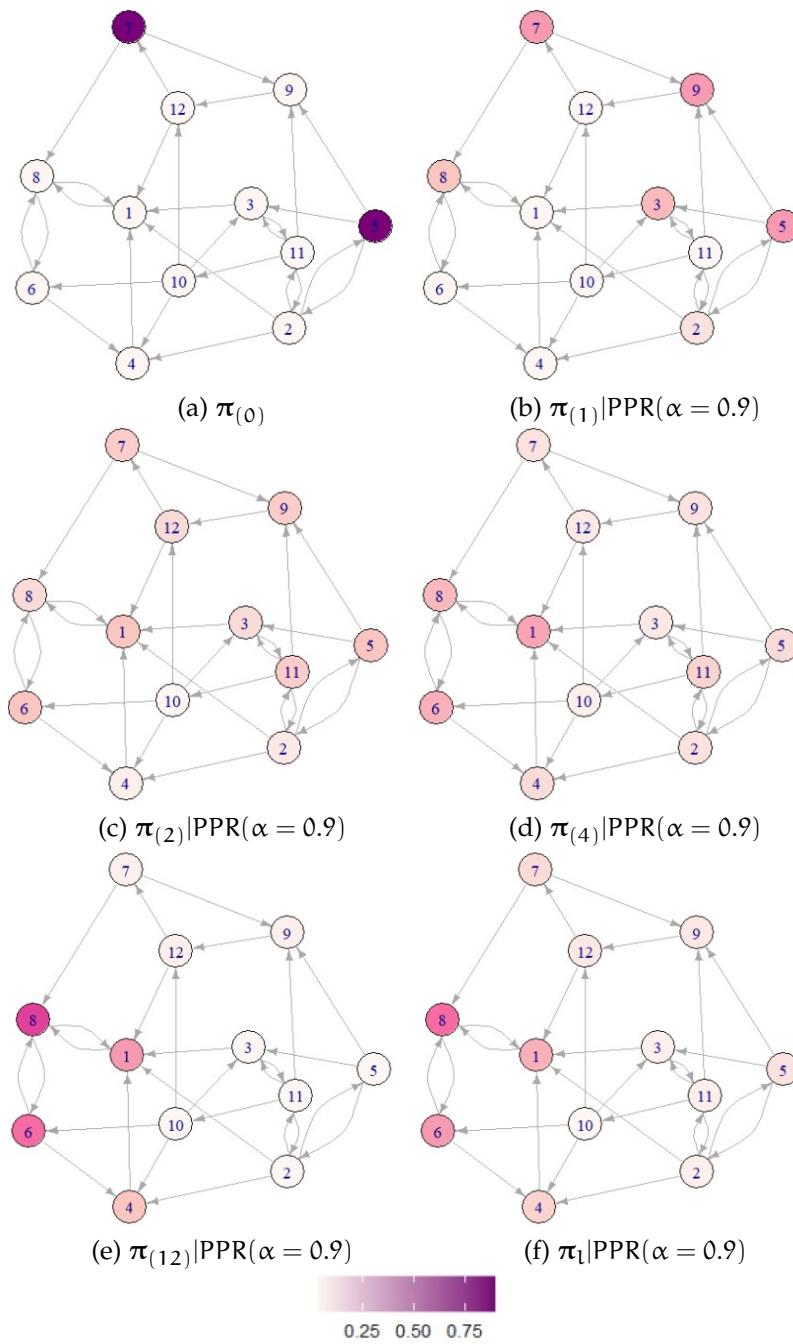


Figure 4.3: PPR convergent probability distribution investigation

We concluded that the PPR diffusion process converged after 12 iterations. The item node '8' had the highest landing probability (0.33) within the vector $\pi_{(12)}$. The linear solution of the PPR kernel exhibited the similar result. The item '8' had the highest landing probability (0.26) within the state vector $\pi_{(1)}$. Thus, item '8' would be the best recommendation to the user using the PPR diffusion process.

4.4.2 Regularized Laplacian Kernel

As a significant part of our research, we present the diffusion process using the Regularized Laplacian kernel. This kernel, a Neumann series-based kernel, follows a strategy similar to the personalized PageRank kernel. Like the PPR kernel, the LAP kernel also has a damping factor (α) to regulate the probability of a random walker moving from the current occupied item node onto one connected item node with the transition probability declared in the inter-item graph. As the probability is close to 1, the random walker will likely move onto a connected item node; otherwise, the random walker jumps back to one of the seed nodes.

We first ran three iterations of the LAP diffusion process and tested the (α) from a set of values 0.1, 0.3, 0.5, 0.7, 0.9. Figure 4.4 below shows the optimal damping factor selection used in the LAP kernel. We observed that the change of the landing probability on the seed item node '7' decreased significantly as the (α) increased from 0.1 to 0.9. Therefore, we selected the ($\alpha = 0.9$) as the optimal damping factor.

Then, we ran the LAP diffusion process with the optimal damping factor on the inter-item graph with three, five, seven, and ten iterations, respectively. Eqs 4.4.7 – 4.4.10 shows the corresponding probability distribution vectors after three, five, seven, and ten iterations. Figure 4.5 (a) shows the initial landing probability distribution of the graph nodes, and Figures 4.5 (b)-(e) show the landing probability distribution of the graph nodes after three, five, seven, and ten iterations. Further, we linearly solved the convergent probability distribution vector in Eq 4.4.11, and show the probability distribution of the graph nodes in Figure 4.5 (f).

We concluded that the diffusion process converged after ten iterations. Item '2' has the highest landing probability within the convergent probability distribution vector. Thus, item '2' would be the best choice for the user using the LAP diffusion process.

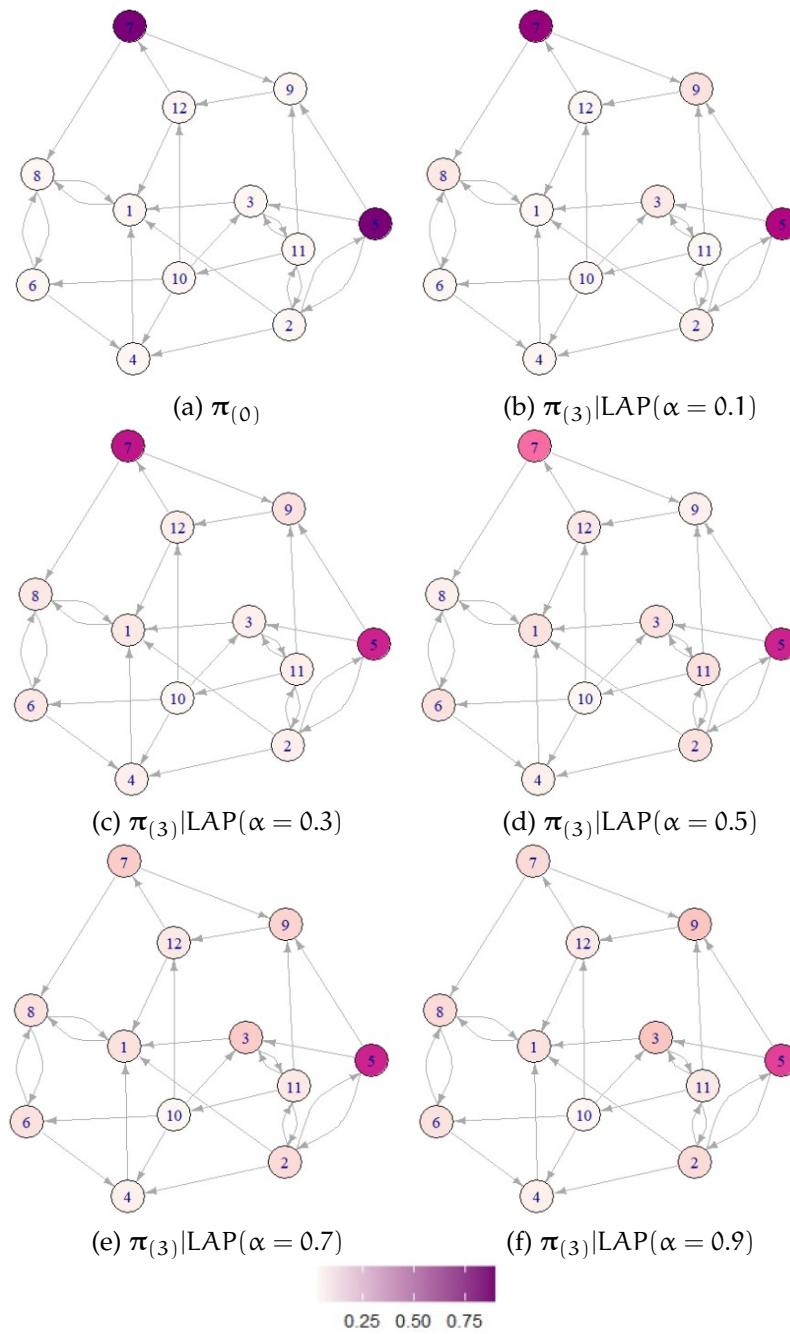
$$\pi_{(3)}^{\text{LAP}(\alpha=0.9)} = \left[0.05 \quad 0.07 \quad 0.13 \quad 0.01 \quad 0.34 \quad 0.05 \quad 0.07 \quad 0.08 \quad 0.13 \quad 0 \quad 0.04 \quad 0.03 \right] \quad (4.4.7)$$

$$\pi_{(5)}^{\text{LAP}(\alpha=0.9)} = \left[0.07 \quad 0.15 \quad 0.13 \quad 0 \quad 0.23 \quad 0.07 \quad 0.01 \quad 0.08 \quad 0.11 \quad 0 \quad 0.1 \quad 0.04 \right] \quad (4.4.8)$$

$$\pi_{(7)}^{\text{LAP}(\alpha=0.9)} = \left[0.07 \quad 0.21 \quad 0.11 \quad 0.01 \quad 0.19 \quad 0.06 \quad 0.02 \quad 0.06 \quad 0.09 \quad 0.01 \quad 0.13 \quad 0.04 \right] \quad (4.4.9)$$

$$\pi_{(10)}^{\text{LAP}(\alpha=0.9)} = \left[0.07 \quad \mathbf{0.26} \quad 0.09 \quad 0.03 \quad 0.17 \quad 0.05 \quad 0.01 \quad 0.05 \quad 0.07 \quad 0.02 \quad 0.15 \quad 0.03 \right] \quad (4.4.10)$$

$$\pi_{(1)}^{\text{LAP}(\alpha=0.9)} = \left[0.05 \quad \mathbf{0.28} \quad 0.07 \quad 0.01 \quad 0.15 \quad 0.03 \quad 0.02 \quad 0.04 \quad 0.06 \quad 0.09 \quad 0.13 \quad 0.04 \right] \quad (4.4.11)$$

Figure 4.4: LAP: damping factor (α) investigation

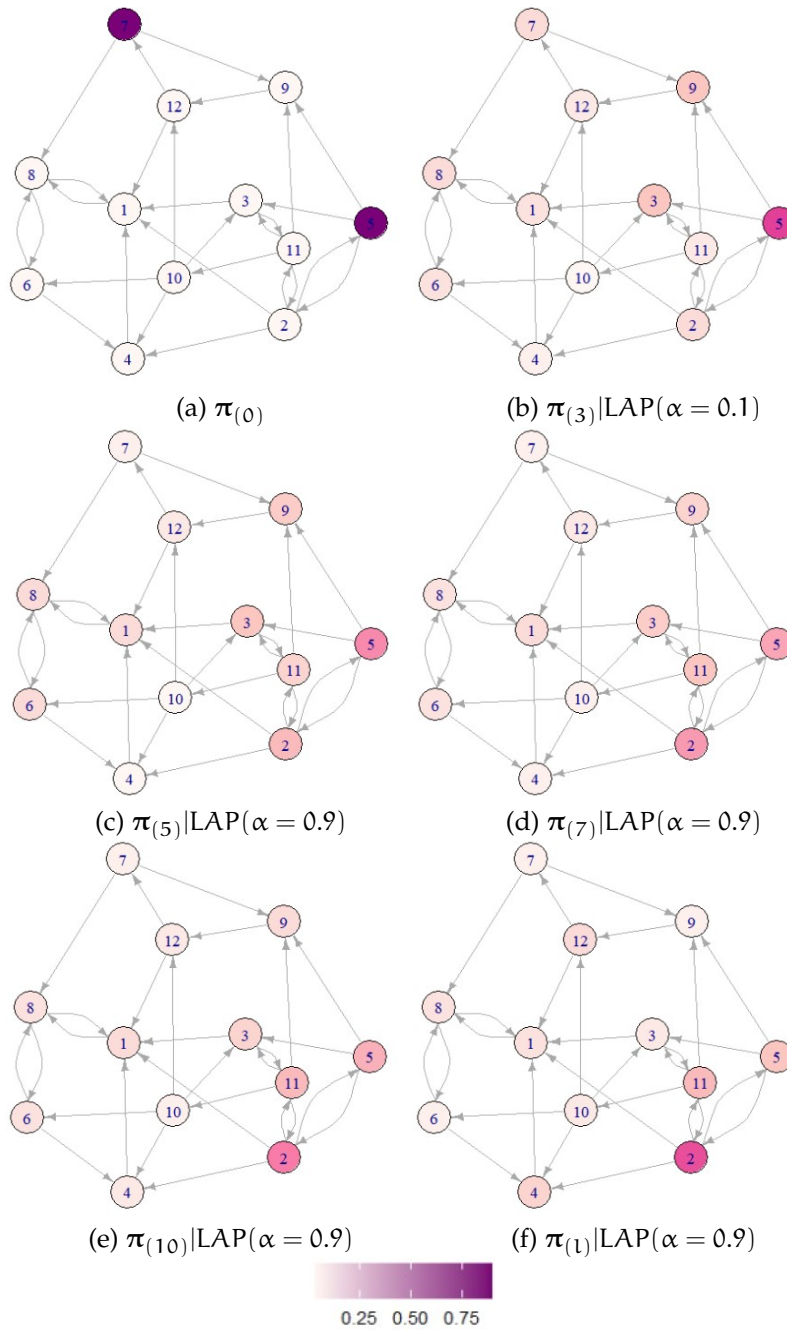


Figure 4.5: LAP convergent probability distribution investigation

4.4.3 Communicability Kernel

Furthermore, we showed the communicability kernel (DR) diffusion process. Compared with the LAP and PPR diffusion kernels, which linear equations can solve, the DR kernel can only be solved using the iterative method. The DR kernel uses a damping factor to normalize the factorial result.

We first tested the damping factor (α) in a set 0.1, 0.5, 1, 5, 10, 20 for three iterative diffusion processes, starting with the seed nodes '5' and '7', as shown in Figure 4.6 below. As the (α) increased, the change of the landing probability on the seed item nodes '5' and '7' decreased significantly. The landing probability remained unchanged as the ($\alpha \geq 10$). Therefore, we chose ($\alpha = 10$) as the optimal damping factor.

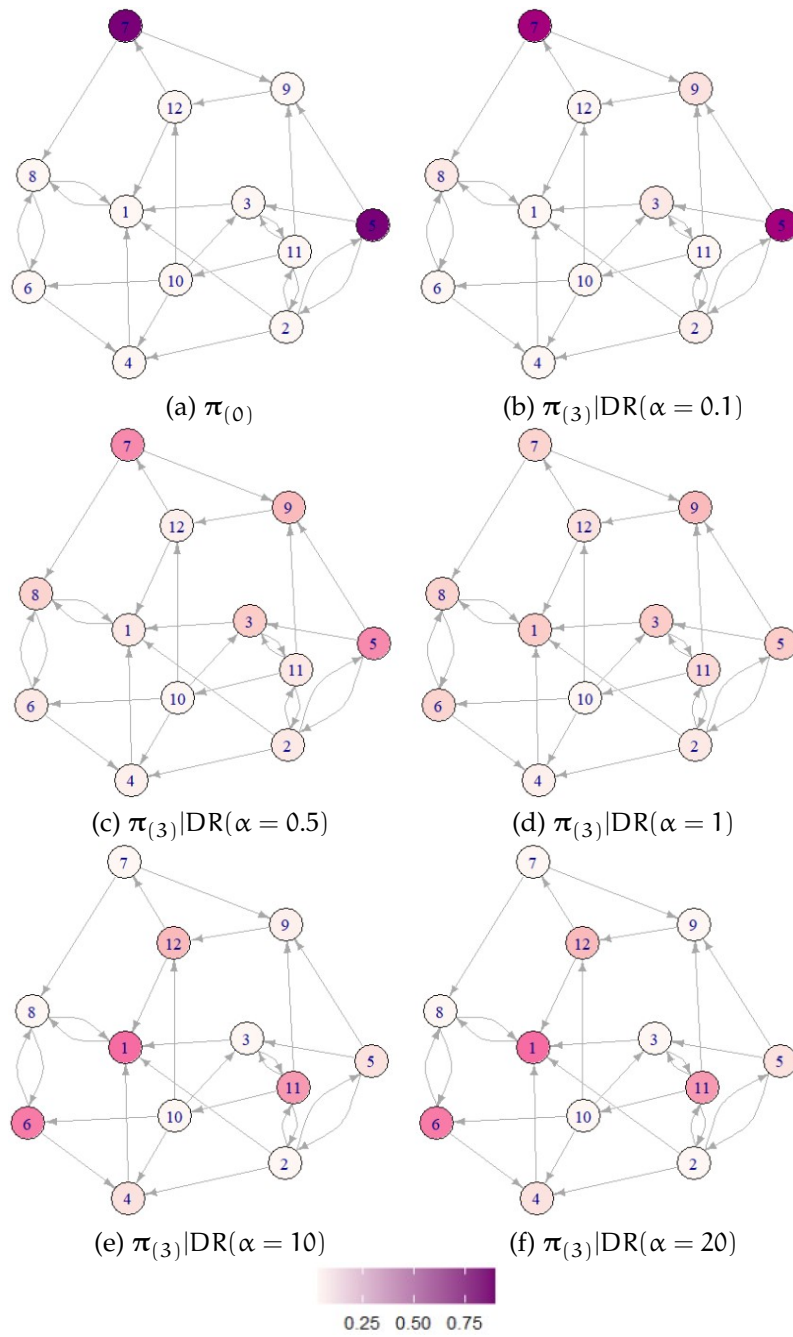
We then run the DR diffusion kernel with ($\alpha = 10$) for two, six, and ten iterations, starting with the seed nodes '5' and '7', and show the results in Figure 4.7 below; the diffusion process reached a convergent state after ten iterations from the initial state, as we can see that the colour of nodes remained unchanged after ten iterations. The node '6' has the highest diffusion score (0.3). Thus, the item '6' would be recommended to the user. Eqs 4.4.12 - 4.4.15 show the landing probability distribution vector after each iteration of the diffusion process.

$$\pi_{(2)}^{\text{DR}(\alpha=10)} = \begin{bmatrix} 0.27 & 0 & 0 & 0.05 & 0.06 & 0.25 & 0 & 0 & 0.01 & 0 & 0.21 & 0.15 \end{bmatrix} \quad (4.4.12)$$

$$\pi_{(6)}^{\text{DR}(\alpha=10)} = \begin{bmatrix} 0.26 & 0 & 0 & 0.13 & 0.06 & 0.27 & 0 & 0.13 & 0 & 0 & 0.11 & 0.03 \end{bmatrix} \quad (4.4.13)$$

$$\pi_{(10)}^{\text{DR}(\alpha=10)} = \begin{bmatrix} 0.25 & 0 & 0 & 0.13 & 0.04 & \mathbf{0.3} & 0 & 0.18 & 0 & 0 & 0.07 & 0.02 \end{bmatrix} \quad (4.4.14)$$

$$\pi_{(20)}^{\text{DR}(\alpha=10)} = \begin{bmatrix} 0.24 & 0 & 0 & 0.15 & 0.02 & 0.3 & 0 & 0.22 & 0 & 0 & 0.03 & 0 \end{bmatrix} \quad (4.4.15)$$

Figure 4.6: DR: damping factor (α) investigation

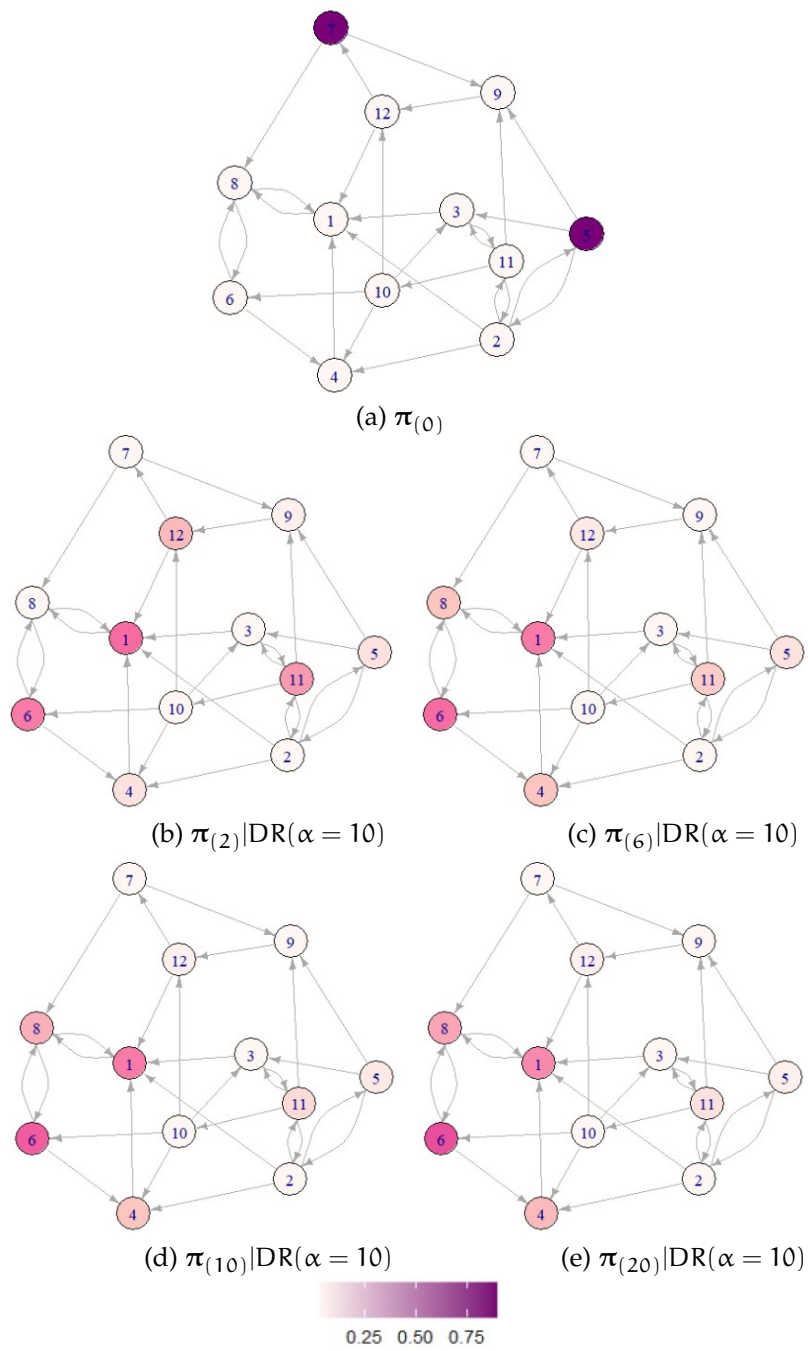


Figure 4.7: DR convergent probability distribution investigation

4.5 CONCLUSION

Overall, this chapter mainly discussed the diffusion kernels. We first introduced the concept of a diffusion kernel and illustrated the diffusion process as an iterative information propagation mechanism over the net. Then, we pointed out the limitation that the SPR kernel only delivered a global recommendation result to the user. That stimulates the requirement of the personalized recommendation method. Next, we explained how a diffusion kernel works with a random surfer on the inter-item graph as a kernel-based method to provide personalized recommendation results. The kernel-based method started with the user's rated items and explored the unseen items via random walks on the inter-graph graph.

Subsequently, we explore a range of practical diffusion kernels, including the Neumann series-based kernels (PPR and LAP) and the exponential-based kernel (DR) in sections 4.2 and 4.3. We elucidate the significance of the damping factor in each type of kernel. For instance, the damping factor in the Neumann series-based kernels represents the probability of a random walker choosing to move from the currently occupied item node to another connected item node following the out-coming link within the inter-item graph or a probability of jumping back to one of the seed nodes. In contrast, the damping factor in the exponential-based kernel serves as a scaling factor to re-scale the landing probability distribution vectors. By illustrating the diffusion process for each diffusion kernel on the actual inter-item graph in section 4.4, we demonstrate how the damping factor in different kernels influences the diffusion results.

Although the kernel-based approach effectively recommends items to users, it ignores the collaborative information between users and items during diffusion. In the next chapter, we will outline our methodology by proposing an integrated framework, 'RecWalk*', incorporating the user-item bipartite graph and an inter-item graph as a completed graph considering the inter-user, inter-item, and user-item relationships. We will also incorporate this framework with a diffusion kernel and investigate how this framework contributes to the recommendation result.

5

METHODOLOGY: GENERATING RECOMMENDATIONS USING GRAPH DIFFUSION KERNELS

In Chapter 4, we have introduced the different diffusion-kernel-based recommendation algorithms for a single user and saw that the user would receive different personalized recommendation results based on the selection of kernels. In this type of method, a random walker only starts with the seed item nodes rated by the user on the inter-item graph and traverses onto unseen item nodes with different transition probabilities. In other words, the recommendations to the users are independent. Such a method ignores the inter-user relationships.

Therefore, we will introduce a random walk framework, 'RecWalk*' [ZH22], incorporating the user-item bipartite graph and an inter-item graph as a complete component considering the inter-user, inter-item, and user-item relationships. Users can personalize their recommendation results via random walks by choosing different diffusion kernels. This framework was initially argued by Karypis and Nikolakopoulos [NK19] in 2019, and we extended this framework with a variety of diffusion kernels.

5.1 MODEL CONSTRUCTION

Next, we will introduce the construction of RecWalk*. As we discussed in Chapter 2, we start with a user-item interaction matrix in the collaborative filtering task. We use the implicit feedback data to fill the user-item matrix, where each non-zero (one) entry represents an interaction a user gives to an item; otherwise, zeros. In other words, the implicit data assumes that the user has the same preference for each rated item.

Table 5.1 below shows an example of the user-item matrix with implicit feedback consisting of five users and ten items with 15 interactions in total. Each user and item in this table starts with the identifiers 'U' and 'I'. Figure 5.1 shows the implicit bipartite user-item graph where the orange nodes are users and the green nodes are items.

The RecWalk* framework consists of two core components: the user-item interaction graph and the inter-item similarity graph. The item nodes connect these two components as a combination graph. Within this combination graph, a random walker can traverse between the user-item component or the inter-item component.

	I1	I2	I3	I4	I5	I6	I7	I8	I9	I10
U1	1	0	0	1	0	0	1	0	0	0
U2	0	1	0	0	0	1	0	0	1	0
U3	0	0	1	1	0	0	0	1	0	1
U4	1	0	0	0	1	0	0	0	0	0
U5	0	0	1	0	0	1	0	0	1	0

Table 5.1: Implicit User-item matrix

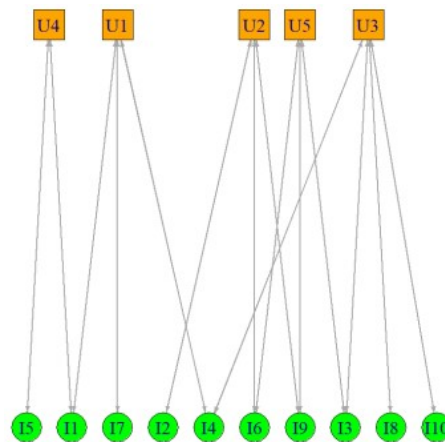


Figure 5.1: Implicit bipartite user-item graph

In chapter 4, we already introduced diffusion kernels and demonstrated how the kernel-based approach generated recommendation results on the inter-item graph, starting with the seed item nodes rated by the user. Within an inter-item graph, a random walker moves from one item node to another node following their transition probability. Therefore, we first declare the transition probability in the user-item bipartite graph used in the RecWalk* model, which is the probability of a walker traversing from a user node to an item node or from an item node to a user node. The RecWalk* model simplifies the weighting scheme in the user-item graph by using the reciprocal value of the number of items the user rated as the weight from a user node to an item node and from an item node to a user node.

Figure 5.2 illustrates the user-item graph weighting scheme in the RecWalk* model, using the users' 'U1' and 'U4' from the original user-item bipartite graph (as shown in Figure 5.1). To simplify the graph representation, we extract the local subgraph from the primary user-item bipartite graph consisting of the users' 'U1' and 'U4' and their all rated items. For example, the user 'U1' with three rated items ('I1', 'I4', and 'I7') has a transition probability of 0.33 to each item node; the user 'U4' with two rated items ('I1', 'I5') has a transition probability of 0.5 to each item node. Similarly, the items 'I4', 'I5', and 'I7' have the same transition probability to the user 'U1', and the items 'I1' and 'I5' have the same transition probability to the 'U4'. Thus, we declare the user-item transition probability matrix as ($\mathbf{H}_{\text{RecWalk}^*}$) in the RecWalk* model.

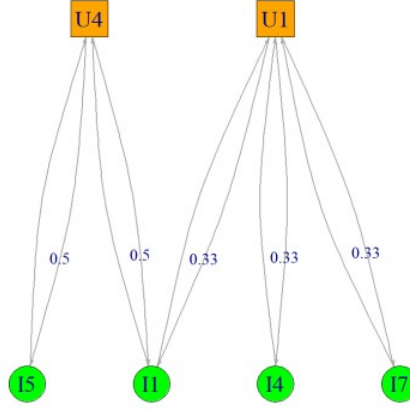


Figure 5.2: RecWalk*: User-item graph weighting scheme

$$\mathbf{A}_G = \begin{bmatrix} 0 & \mathbf{R} \\ \mathbf{R}^\top & 0 \end{bmatrix} \quad (5.1.1)$$

$$\mathbf{H}_{\text{RecWalk}^*} = \begin{bmatrix} 0 & \mathbf{R}_x \\ \mathbf{R}_x^\top & 0 \end{bmatrix}, \mathbf{R}_x[i, j] = \frac{\mathbf{R}[i, j]}{\mathbf{R}\mathbf{1}[i, j]} \quad (5.1.2)$$

Eq 5.1.1 denotes the adjacency matrix of the bipartite graph (\mathbf{A}_G); Eq 5.1.1 defines the transition probability matrix ($\mathbf{H}_{\text{RecWalk}^*}$) of the user-item component of the RecWalk* model. Within the declaration of the matrix (\mathbf{R}_x) in Eq 5.1.2, \mathbf{R} is the implicit user-item interaction matrix and $\mathbf{1}$ is a column-wise vector that fills in one, which has the same dimensionality as the number of users in the database. The matrix (\mathbf{R}_x) is denoted as the element-wise division of the matrices (\mathbf{R}) and ($\mathbf{R}\mathbf{1}$). Within the Eq 5.1.2, the index $[i, j]$ represents the element in the i_{th} row and the j_{th} column in the matrix.

Subsequently, we move on to the declaration of the inter-item component of the RecWalk* model. There are several ways to construct the inter-item component. For instance, we can build an inter-item similarity matrix using the cosine similarity measure. Then, the user-item similarity matrix is used as the adjacency matrix to construct an inter-item similarity graph, as depicted in Figure 5.3 below.

Within the inter-item graph of the RecWalk model, Karypis and Nikolakopoulos [NK19] used the node similarity for a pair of items as the transition probability. In addition, they also added a constant value to each item node so that a walker can stay put on the currently occupied item node with probability. Eq 5.1.3 defines the inter-item component's transition probability matrix (\mathbf{M}_I). In this equation, (\mathbf{W}) is the inter-item similarity matrix corresponding to an item node's transition probability to all other connected item nodes. Karypis and Nikolakopoulos used the matrix ($\|\mathbf{W}\|_\infty$) to rescale the inter-item similarity matrix. In addition, the term $\text{Diag}(\mathbf{1} - \frac{1}{\|\mathbf{W}\|_\infty} \mathbf{W}\mathbf{1})$ contains the probability that an item node is redirected to itself. The matrix ($\|\mathbf{W}\|_\infty$) represents the maximum value of all row sums of (\mathbf{W}), and $\mathbf{1}$ is a column-wise vector filling of the ones with the size of the number of items. Thus, in the inter-item graph, a random walker can traverse from an item node onto another node following the nodes' similarity as the transition

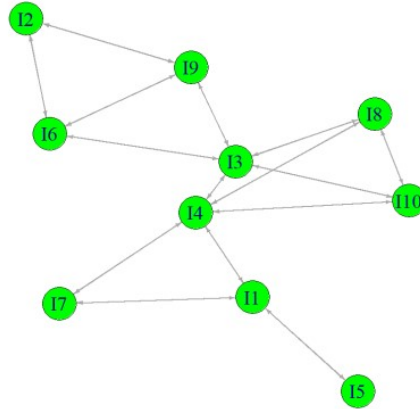


Figure 5.3: Inter-item similarity graph (Cosine)

probability or stay put on the currently occupied item node with a probability. In addition, we also add a self-loop to each user node for all users so that a random walker can stay put at the currently occupied user node with a probability.

$$\mathbf{M}_I = \frac{1}{\|\mathbf{W}\|_\infty} \mathbf{W} + \text{Diag}(\mathbf{1} - \frac{1}{\|\mathbf{W}\|_\infty} \mathbf{W}\mathbf{1}) \quad (5.1.3)$$

We have now defined the transition probability matrices to the user-item and the inter-item components in the RecWalk* framework. We need to find a way to combine these two components so that a random walker can randomly traverse between the user-item and the inter-item components. We adopted the ‘biased coin toss’ strategy, which Karypis and Nikolakopoulos argued, to simulate a random walk on the RecWalk* model. The ‘biased coin toss’ strategy declares a biased random walk that a random walker can choose to move within the user-item component or the inter-item component by a probability decided by the ‘biased coin toss’ process.

Two hypotheses are underlying this strategy: (1) A random walker always starts with a user node and terminates on an item node that is not rated by the user yet; (2) The inter-item graph must be a connected graph to ensure that each item node within the inter-item graph is reachable.

Figure 5.4 below demonstrates the random walk stimulation process within the RecWalk* model. Firstly, Figure 5.4 (a) shows a user-item combination graph where the square orange nodes are users, and the circle green nodes are items. Within the random walk simulation strategy, each move is determined by a biased coin toss. Assuming that the walker currently occupies a node ($c \in U \cup I$), a biased coin toss determines the next possible move that yields heads with probability (θ) or tails with probability ($1-\theta$). Provided that the current node (c) is a user node ($c \in U$): i) if the coin-toss yields heads, the walker jumps to one of the item nodes rated by the user randomly; ii) if the coin-toss yields tails, the walker stays put. While the current node (c) is an item node ($c \in I$): i) if the coin-toss yields heads, the walker jumps to one of the users that have rated the item randomly; ii) if the coin-toss yields tails, the walker randomly moves to one of the item nodes which connects with the currently occupied item node.

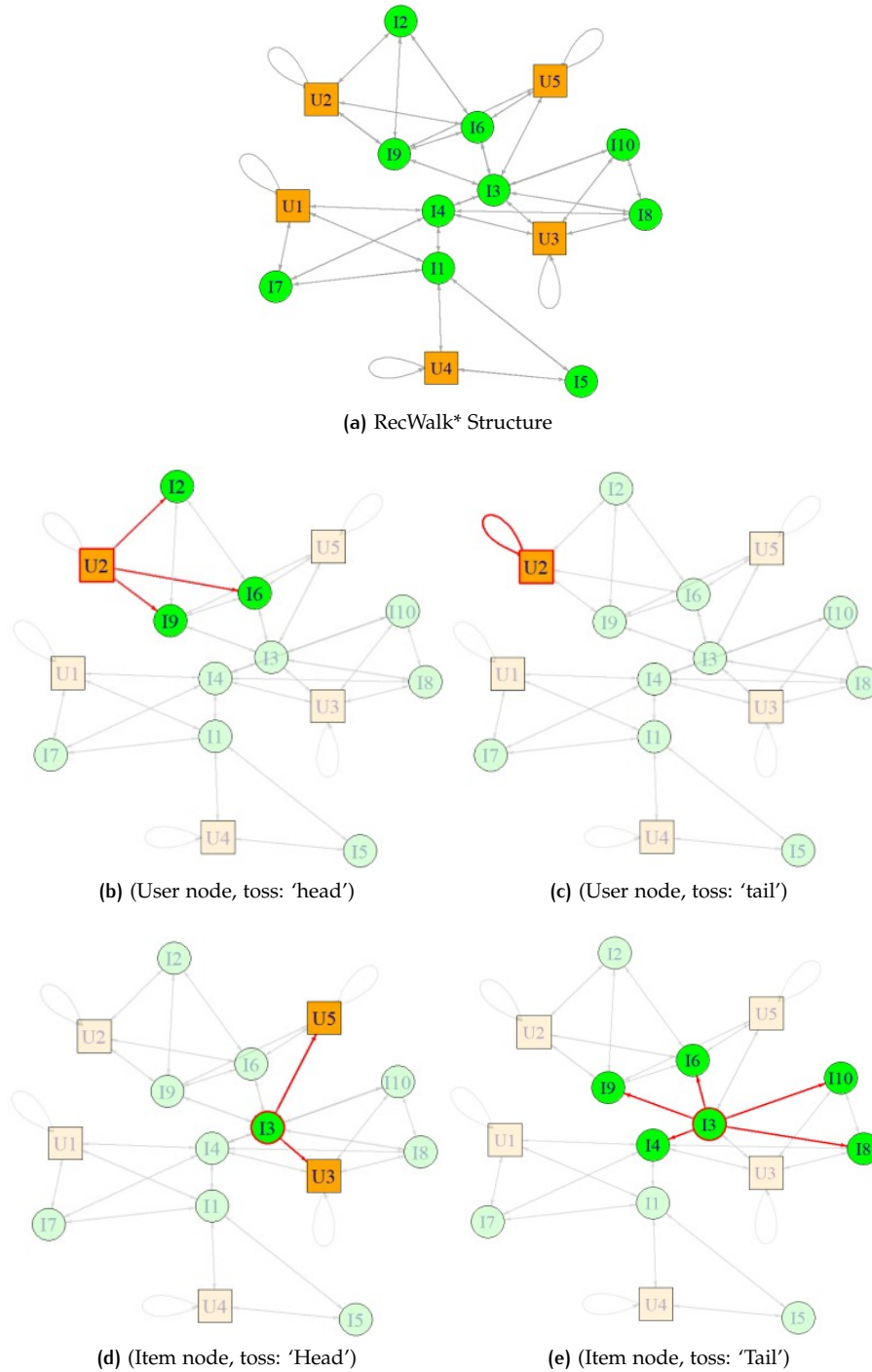


Figure 5.4: RecWalk*: Walk simulation

We provide an example to illustrate one possible move in the RecWalk* model. Suppose tossing a biased coin has a probability of 0.6 that it will go head and a probability of 0.4 that it will go tail. Figures 5.4 (b) and (c) show where the walker occupied the current node as a user node. We selected the user 'U2' as the target user (the square with red borders). If the coin yields on 'head' (as shown in (b)), the walker will move onto one of the following rated items ('I2', 'I6', 'I9') randomly with their transition probability declared in the user-item transition probability matrix ($\mathbf{H}_{\text{RecWalk}^*}$); otherwise, the walker will stay put at the same position (as shown in Figure 5.4 (c)). Alternatively, Figures 5.4 (d) and (e) demonstrate the situation as the currently occupied node by the walker was an item node. We chose the item 'I3' as the target item. If the coin yields on 'head' (as shown in Figure 5.4 (d)), the walker will move on to one of the connected users ('U3', 'U5') randomly with their transition probability defined in the inter-item transition probability matrix ($\mathbf{H}_{\text{RecWalk}^*}$); otherwise, the walker will randomly jump to one of the connected item nodes ('I4', 'I6', 'I8', 'I9', 'I10') with the transition probability defined in the transition probability matrix (\mathbf{M}_I) (as shown in Figure 5.4 (e)).

Furthermore, Algorithm 5.1 implements the RecWalk* model. This algorithm accepts three inputs as parameters: a user-item matrix (\mathbf{R}), an inter-item similarity matrix (\mathbf{W}), and a probability (θ) that connects the user-item and inter-item components generated by the flipping of a biased coin. Line 1 constructs the inter-item transition probability matrix (\mathbf{M}_I), and lines 2-4 construct the user-item transition probability matrix ($\mathbf{H}_{\text{RecWalk}^*}$). Line 5 declares the combination transition probability matrix (\mathbf{P}) that combines the user-item transition probability matrix ($\mathbf{H}_{\text{RecWalk}^*}$) and the inter-item transition probability matrix (\mathbf{M}_I) using the probability (θ).

Algorithm 5.1: RecWalk* Model

Input: user-item matrix \mathbf{R} , inter-item similarity matrix \mathbf{W} , parameter: θ

Output: \mathbf{P}

- 1 Construct \mathbf{M}_I : $\mathbf{M}_I \leftarrow \frac{1}{\|\mathbf{W}\|_\infty} \mathbf{W} + \text{Diag}(\mathbf{1} - \frac{1}{\|\mathbf{W}\|_\infty} \mathbf{W}\mathbf{1})$
 - 2 Construct \mathbf{A}_G : $\mathbf{A}_G \leftarrow \begin{bmatrix} \mathbf{o} & \mathbf{R} \\ \mathbf{R}^\top & \mathbf{o} \end{bmatrix}$
 - 3 $\mathbf{R}_x[i, j] \leftarrow \frac{\mathbf{R}[i, j]}{\mathbf{R}\mathbf{1}[i, j]}$
 - 4 Construct $\mathbf{H}_{\text{RecWalk}^*}$: $\mathbf{H}_{\text{RecWalk}^*} \leftarrow \begin{bmatrix} \mathbf{o} & \mathbf{R}_x \\ \mathbf{R}_x^\top & \mathbf{o} \end{bmatrix}$
 - 5 Construct \mathbf{P} : $\mathbf{P} \leftarrow \theta \mathbf{H}_{\text{RecWalk}^*} + (1 - \theta) \begin{bmatrix} \mathbf{I} & \mathbf{o} \\ \mathbf{o} & \mathbf{M}_I \end{bmatrix}$
-

In short, a random walker starts at a user node and can move within the user-item component or the inter-item component following a transition probability or stay put on the node it currently occupies. The terminal node of a random walker is always an item node which the user has not rated. We execute the random walks multiple times for a user on the user-item combination graph and return the item nodes that have not been visited by the user and have the highest landing probabilities. As introduced in chapter 4, we have seen that the user can receive personalized recommendation results via running random walks on the inter-item graph based on the selection of different diffusion kernels. Similarly, we can integrate the diffusion kernels onto the user-item combination graph, which motivates us to investigate the recommendation results. In

the next section, we will illustrate how a diffusion kernel cooperates with the RecWalk* model and makes personalized recommendations to the user via random walks.

5.2 MODEL INSTANTIATION

This section emphasizes how the RecWalk* model works with different diffusion kernels to make users personalized recommendations. Sections 5.2.1 and 5.2.2 focus on two Neumann-series-based kernels: the regularized laplacian kernel (LAP) and the personalized PageRank kernel (PPR); Section 5.2.3 shows the exponential kernel, the communicability kernel (DR). We use a landing probability distribution vector (π_u) to record the diffusion state for each iteration.

We declare a one-hot row-wise vector \mathbf{e}_u^\top with the size of $(|U| + |I|)$ that represents the initial state of the user (u) in the RecWalk* model. The vector \mathbf{e}_u^\top contains element 1 in the position that corresponds to user (u) and zeros elsewhere. We use the user 'U2' as an example to illustrate the initial landing probability distribution vector $\pi_{(0)}$ as shown in Eq 5.2.1. In this equation, the entry corresponding to the user 'U2' is '1', and all other entries are zeroes. We used the user 'U2' as the target user to demonstrate the diffusion process working with the RecWalk* model. In this example, we adopted cosine to construct the inter-item model (\mathbf{W}).

$$\mathbf{e}_u^\top = \pi_{(0)} = \left[0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \mid 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \right] \quad (5.2.1)$$

5.2.1 RecWalk* + PPR

Firstly, Algorithm 5.2 below illustrates the RecWalk* model with the PPR kernel. The algorithm receives four parameters: a target user (u), a damping factor (α) associated with the diffusion kernel, a step number (k) indicating the number of iterations performed by the diffusion process, and the combination transition probability matrix (\mathbf{P}) of the RecWalk* model (as described in Algorithm 1). This algorithm outputs a convergent landing probability distribution vector $\pi_{(u)}$ to the user u .

In this algorithm, the first line initializes the starting state $\pi_{(0)}$ to the user (u) with \mathbf{e}_u^\top . Lines 2-7 implement the PPR kernel in the 'Truncated' mode in an iterative way, and the landing probability distribution vector $\pi_{(t)}$ represents the diffusion state after (t) iterations and the result is scaled via the L1-normalization (Line 6). Lines 9-11 illustrate the linear solution of the convergent landing probability distribution vector of RecWalk*+ PPR in the 'Linear' mode.

As the RecWalk* + PPR algorithm accepts four parameters, we first investigate the optimal (θ) of the combination transition probability matrix (\mathbf{P}) that connects the user-item component and inter-item component in the RecWalk* model; then, we try to figure out the optimal damping factor (α) used in the PPR kernel and the optimal number of steps (k) of a random walk.

Algorithm 5.2: RecWalk*+PPR

Input: a user $u \in \mathcal{U}$, a damping factor: α , step number: k , a combination transition probability matrix $\mathbf{P} : \mathbf{P} \leftarrow \text{RecWalk}^*(\mathbf{R}, \mathbf{W}, \theta)$

Output: a convergent landing probability distribution vector $\boldsymbol{\pi}_{(u)}$

```

1 Initialize  $\boldsymbol{\pi}_{(0)} : \boldsymbol{\pi}_{(0)} \leftarrow \mathbf{e}_u$ 
2 if mode == 'Truncated' then
3    $t \leftarrow 1$ 
4   while  $t \leq k$  do
5      $\boldsymbol{\pi}_{(t)} \leftarrow \alpha \boldsymbol{\pi}_{(t-1)} \mathbf{P} + \boldsymbol{\pi}_{(0)}$ 
6     Normalize  $\boldsymbol{\pi}_{(t)} : \boldsymbol{\pi}_{(t)} \leftarrow \frac{\boldsymbol{\pi}_{(t)}}{\|\boldsymbol{\pi}_{(t)}\|}$ 
7   end
8   Assign  $\boldsymbol{\pi}_{(u)} : \boldsymbol{\pi}_{(u)} \leftarrow \boldsymbol{\pi}_{(t)}$ 
9 end
10 else if mode == 'Linear' then
11    $\boldsymbol{\pi}_{(u)} \leftarrow \boldsymbol{\pi}_{(0)} [\mathbf{I} - \alpha \mathbf{P}]^{-1}$ 
12   Normalize  $\boldsymbol{\pi}_{(t)} : \boldsymbol{\pi}_{(t)} \leftarrow \frac{\boldsymbol{\pi}_{(t)}}{\|\boldsymbol{\pi}_{(t)}\|}$ 
13 end

```

We adopted the 'control variates' strategy [SS15] to explore the parameters θ, α , and k . The idea of this strategy means that we only make one parameter as a variable and freeze the rest of the parameters as constant values in experiments.

Firstly, we randomly selected ($\alpha = 0.5$) and ($k = 5$) as the fixed parameters for the damping factor and the step number in the PPR diffusion kernel to observe (θ) from a set of values: 0.005, 0.1, 0.5, 0.7, 0.9, 0.95, and recorded the landing probability distribution vector for each iteration in eqs.(4.6.2) - (4.6.6) below. In these equations, we only recorded the landing probability on the user node 'U2'. However, we neglected the landing probabilities for all the rest of the nodes as we mainly wanted to find the maximum change of the landing probability on that user node starting with the initial probability '1'. As the (θ) increased from 0.005 to 0.95, the change of the landing probability on the user node 'U2' increased until ($\theta = 0.95$). Thus, the change of the landing probability on the user node 'U2' reached the maximum value as ($\theta = 0.95$), and we selected 0.95 as the optimal parameter (θ) in the combination transition probability of the RecWalk* model.

$$\boldsymbol{\pi}_{(5)}^{(\theta=0.005)} | \text{PPR}(\alpha = 0.5) = \left[\dots (0.99)_{U2} \dots \right] \quad (5.2.2)$$

$$\boldsymbol{\pi}_{(5)}^{(\theta=0.1)} | \text{PPR}(\alpha = 0.5) = \left[\dots (0.91)_{U2} \dots \right] \quad (5.2.3)$$

$$\boldsymbol{\pi}_{(5)}^{(\theta=0.5)} | \text{PPR}(\alpha = 0.5) = \left[\dots (0.72)_{U2} \dots \right] \quad (5.2.4)$$

$$\pi_{(5)}^{(\theta=0.7)} | \text{PPR}(\alpha = 0.5) = [\dots (0.66)_{U2} \dots] \quad (5.2.5)$$

$$\pi_{(5)}^{(\theta=0.9)} | \text{PPR}(\alpha = 0.5) = [\dots (0.62)_{U2} \dots] \quad (5.2.6)$$

Next, we used the $(\theta = 0.95, k = 5)$ as the fixed parameters to test the PPR diffusion damping factor (α) . As the control variates approach used before, we run the (α) from values 0.1, 0.3, 0.5, 0.7, 0.9 to find the optimal damping factor. Eqs 5.2.2 – 5.2.6 show the change of the landing probability on the user node 'U2' after five iterations (The dots mean the omitted items). The change of the landing probability on the user node 'U2' is increasing significantly as the (α) increased from 0.1 to 0.9. Therefore, we used $(\alpha = 0.9)$ as the optimal PPR diffusion damping factor, as the kernel has the highest transmission efficiency. In other words, one node is easily transmitted to other nodes quickly on the graph.

$$\pi_{(5)}^{(\theta=0.95)} | \text{PPR}(\alpha = 0.1) = [\dots (0.99)_{U2} \dots] \quad (5.2.7)$$

$$\pi_{(5)}^{(\theta=0.95)} | \text{PPR}(\alpha = 0.3) = [\dots (0.75)_{U2} \dots] \quad (5.2.8)$$

$$\pi_{(5)}^{(\theta=0.95)} | \text{PPR}(\alpha = 0.5) = [\dots (0.61)_{U2} \dots] \quad (5.2.9)$$

$$\pi_{(5)}^{(\theta=0.95)} | \text{PPR}(\alpha = 0.7) = [\dots (0.48)_{U2} \dots] \quad (5.2.10)$$

$$\pi_{(5)}^{(\theta=0.95)} | \text{PPR}(\alpha = 0.9) = [\dots (0.35)_{U2} \dots] \quad (5.2.11)$$

From now on, we have obtained the two optimal parameters $(\alpha = 0.95)$ and $(\alpha = 0.9)$ for the RecWalk*+PPR algorithm. Finally, we only tested the random walks with a different number of steps (k) from 5, 10, 15, 20 on the graph to observe the landing probabilities on all unrated item nodes as the diffusion process is going to be converged. The items with highest landing probabilities would be returned to the user as the recommendation result. Figures 5.5 (a) - (f) demonstrate the landing probabilities after 5, 10, 15, and 20 iterations for seven items ('I1', 'I3', 'I4', 'I5', 'I7', 'I8', 'I10') the user 'U2' has not rated yet, and neglects all items the user 'U2' has rated (the nodes are marked as gray). Eqs.(5.2.12) – (5.2.15) gives the landing probability distribution vector on seven unrated items ('I1', 'I3', 'I4', 'I5', 'I7', 'I8', 'I10') to the user 'U2'.

With the increasing value of (k) , the diffusion process converged after 20 iterations ($k = 20$). Thus, item 'I3' (0.52) was the best candidate recommendation to the user 'U2'; the item 'I4' (0.15) was the second best choice to the user 'U2'; and items 'I8' and 'I10' (0.12) were both the third options to the user 'U2'.

$$\pi_{(5)}^{(\theta=0.95)} | \text{PPR}(\alpha = 0.9) = [\dots (0)_{I1} \quad (0.80)_{I3} \quad (0.06)_{I4} \quad (0)_{I5} \quad (0)_{I7} \quad (0.07)_{I8} \quad (0.07)_{I10} \dots] \quad (5.2.12)$$

$$\pi_{(10)}^{(\theta=0.95)} | \text{PPR}(\alpha = 0.9) = [\dots (0.02)_{I1} \quad (0.62)_{I3} \quad (0.12)_{I4} \quad (0)_{I5} \quad (0.12)_{I7} \quad (0.11)_{I8} \quad (0.11)_{I10} \dots] \quad (5.2.13)$$

$$\pi_{(15)}^{(\theta=0.95)} | \text{PPR}(\alpha = 0.9) = [\dots (0.04)_{I1} \quad (0.55)_{I3} \quad (0.14)_{I4} \quad (0)_{I5} \quad (0.03)_{I7} \quad (0.12)_{I8} \quad (0.12)_{I10} \dots] \quad (5.2.14)$$

$$\pi_{(20)}^{(\theta=0.95)} | \text{PPR}(\alpha = 0.9) = [\dots (0.05)_{I1} \quad (0.52)_{I3} \quad (0.15)_{I4} \quad (0.01)_{I5} \quad (0.03)_{I7} \quad (0.12)_{I8} \quad (0.12)_{I10} \dots] \quad (5.2.15)$$

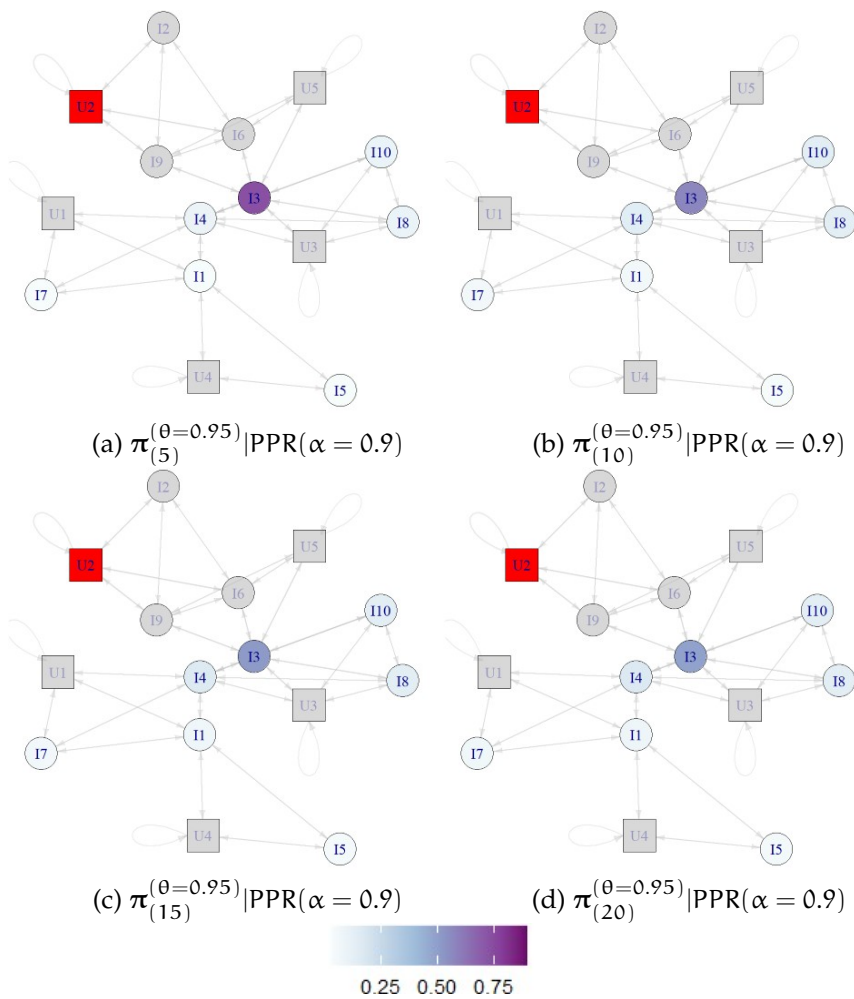


Figure 5.5: RecWalk*: $\pi_{(t)}^{(\theta=0.95)} | \text{PPR}(\alpha = 0.9)$

5.2.2 RecWalk* + LAP

As LAP is another Neumann series-based diffusion kernel, we present the RecWalk*+LAP model in Algorithm 5.3 below. The input and initialization procedures are the same as those used for the RecWalk*+PPR model. Compared with the RecWalk*+PPR model, RecWalk* + LAP uses the Laplacian matrix (\mathbf{L}) of the user-item combination graph as the combination transition probability matrix, as shown in line 3. Lines 4-10 and 11-14 give the 'Truncated' and 'Linear' update equations of the landing probability vectors for users and items, respectively. L1-normalization scales the landing probability vector after each iteration for the 'Truncated' way until it converges, as shown in Line 8; the convergent landing probability vector is scaled by the L1-normalization for the 'Linear' way, as shown in Line 13.

Algorithm 5.3: RecWalk*+LAP

Input: a user $u \in \mathcal{U}$, a damping factor: α , step number: k , a combination transition probability matrix $\mathbf{P} : \mathbf{P} \leftarrow \text{RecWalk}^*(\mathbf{R}, \mathbf{W}, \theta)$

Output: a convergent landing probability distribution vector $\pi_{(u)}$

```

1 Initialize  $\pi_{(0)} : \pi_{(0)} \leftarrow \mathbf{e}_u$ 
2  $\mathbf{D} \leftarrow \text{Diag}(\mathbf{P}\mathbf{1})$ 
3  $\mathbf{L} \leftarrow \mathbf{D} - \mathbf{P}$ 
4 if mode == 'Truncated' then
5    $t \leftarrow 1$ 
6   while  $t \leq k$  do
7      $\pi_{(t)} \leftarrow \alpha \pi_{(t-1)} \mathbf{L} + \pi_{(0)}$ 
8     Normalize  $\pi_{(t)} : \pi_{(t)} \leftarrow \frac{\pi_{(t)}}{\|\pi_{(t)}\|}$ 
9   end
10  Assign  $\pi_{(u)} : \pi_{(u)} \leftarrow \pi_{(t)}$ 
11 end
12 else if mode == 'Linear' then
13    $\pi_{(u)} \leftarrow \pi_{(0)} [\mathbf{I} - \alpha \mathbf{L}]^{-1}$ 
14   Normalize  $\pi_{(t)} : \pi_{(t)} \leftarrow \frac{\pi_{(t)}}{\|\pi_{(t)}\|}$ 
15 end

```

5.2.3 RecWalk* + DR

Furthermore, we can combine the RecWalk* model with an exponential kernel like the communicability kernel (DR). Algorithm 5.4 below implements the RecWalk*+DR kernel in detail. Compared with the Neumann-series-based kernels, as discussed in sections 5.2.1 and 5.2.2, such kernel uses the 'Truncated' way only. The algorithm accepts four variables as the inputs. The first line is the initialization procedure which is the same as those used in RecWalk*+PPR and RecWalk*+LAP. Lines 2-6 are the iterative update equation of the landing probability distribution vector based on the Eq 4.3.2. After each iteration of the diffusion process, the landing probability distribution vector is scaled by L1-Normalization, as shown in line 4. At the end of this algorithm, it returns $\pi_{(u)}$ as the final landing probability distribution vector to the user (u) in line 6.

Algorithm 5.4: RecWalk* + DR

Input: a user $u \in \mathcal{U}$, a damping factor: α , step number: k , a combination transition probability matrix $\mathbf{P} : \mathbf{P} \leftarrow \text{RecWalk}_*(\mathbf{R}, \mathbf{W}, \theta)$

Output: a convergent landing probability distribution vector $\boldsymbol{\pi}_{(u)}$

```

1 Initialize  $\boldsymbol{\pi}_{(0)} : \boldsymbol{\pi}_{(0)} \leftarrow \mathbf{e}_u$ 
2  $t \leftarrow 1$ 
3 while  $t \leq k$  do
4    $\boldsymbol{\pi}_{(t)} \leftarrow \boldsymbol{\pi}_{(t-1)} + \boldsymbol{\pi}_{(0)} \alpha^t \frac{\mathbf{P}^t}{t!}$ 
5   Normalize  $\boldsymbol{\pi}_{(t)} : \boldsymbol{\pi}_{(t)} \leftarrow \frac{\boldsymbol{\pi}_{(t)}}{\|\boldsymbol{\pi}_{(t)}\|}$ 
6 end
7 Assign  $\boldsymbol{\pi}_{(u)} : \boldsymbol{\pi}_{(u)} \leftarrow \boldsymbol{\pi}_{(t)}$ 

```

5.3 CONCLUSION

Overall, this chapter primarily gives a systematic introduction to RecWalk*. This graph-based random walk framework incorporates a diffusion kernel personalized by the user (introduced in Chapter 4) as an effective recommender method. Firstly, in section 5.1, we mainly introduced the concept of the RecWalk* framework using graph demonstrations to simulate a random surfer as a user or item on the graph. Then, we implement the RecWalk* framework with different diffusion kernels (as introduced in sections 4.2 and 4.3) in section 5.2. We used the PPR kernel as an example to demonstrate how we applied this framework with a diffusion kernel to generate a recommendation result demonstrated using the landing probability vectors and graphs.

In the next chapter, we will set up and detail our experimental settings and procedures and examine how to evaluate the kernel-based recommendation method.

6

EVALUATION OF GRAPH DIFFUSION KERNEL-BASED RECOMMENDATIONS

This chapter mainly discusses the experimental settings and evaluation procedure of kernel-based recommendation approaches. In section 6.1, we first introduce eight well-known public standard recommendation datasets. Then, we list some non-item and item-based classical CF recommendation algorithms as baseline algorithms and some diffusion kernels combined with the RecWalk* model in section 6.2. Next, we set up our experimental procedures and evaluation metrics in section 6.3 and conduct the experiments with baseline algorithms and the RecWalk* model on eight standard recommendation datasets, showing and comparing their experimental results.

6.1 DATA

In this thesis, we first introduce the common standard recommendation datasets. We selected eight well-known rating datasets from different domains. Most provide explicit feedback (e.g., ratings), and one only provides implicit feedback (e.g., purchasing records).

The Movielens!1M (ML-1M) [HK15] and Yahoo!Movie (YM) [GFS+09] datasets represent film ratings. The Amazon product datasets, provided by McAuley in 2014 [HM16], represent consumer ratings of products: Baby (AM-BY), Cell Phones and Accessories (AM-CP), Apps for Android (AM-AP), and Health and Care (AM-HC). Book-crossing (BX) (explicit rating version), collected by Ziegler [ZMK+05], represents the user's ratings of books. Steam Video Game (SEM) represents users' purchasing records from a popular PC Gaming hub [Rom22].

We applied some filtering to each dataset (except for ML-1M) as the sparse users and items caused problems when sampling data for training and testing. Our filtering approach ensures that each user has at least three rated items and at least one user rates each item.

Table 6.1 presents a comparison of the unfiltered and filtered data. For each dataset, we calculated the number of users (#User), the number of items (#Item), the total number of user-item interactions (N), and the dataset density (ρ) for both unfiltered and filtered versions. The density level is defined as the ratio of N and the user-item rating matrix's size (#User \times #Item). Notably, we did not filter the ML-1M dataset to verify the accuracy and correctness of our baseline algorithms by reproducing the experimental results and comparing them with others' works.

As shown in Table 6.1, the users and items in each filtered dataset are the subsets of the unfiltered dataset, and the density level increased significantly. Our experiments treat all explicit ratings

Dataset		#User	#Item	N	ρ
ML-1M	Unfiltered	6040	3706	1,000,209	4.47%
	Filtered	6040	3076	1,000,209	4.47%
YM	Unfiltered	7642	11,916	221,367	0.24%
	Filtered	7613	3787	207,747	0.72%
AM-BY	Unfiltered	531,800	64,426	915,446	0.0027%
	Filtered	4266	7009	64,690	0.22%
AM-HC	Unfiltered	1,851,132	252,331	2,982,326	0.0006%
	Filtered	3615	8214	79,757	0.27%
AM-AP	Unfiltered	1,323,884	61,275	2,638,172	0.0032%
	Filtered	3297	7040	106,985	0.46%
AM-CP	Unfiltered	2,261,045	319,678	3,447,249	0.0005%
	Filtered	5142	7916	55,657	0.14%
BX	Unfiltered	7642	11,916	221,367	0.24%
	Filtered	3716	7256	85,370	0.32%
SEM	Unfiltered	12,393	5155	129,511	0.20%
	Filtered	5218	5124	120,854	0.44%

Parameters Settings: #User and #Item: The number of users and items respectively. N: The number of interactions between users and items. ρ : The density level: $N/(\#User \times \#Item)$

Table 6.1: Statistics of eight standard recommendation datasets

as implicit ratings with values of one or zero. In the next section, we will discuss about the algorithms including the baseline algorithms and kernel-based RecWalk* algorithm.

6.2 ALGORITHMS

This section mainly talks about the recommendation algorithms used in this thesis. Section 6.2.1 includes the non-item baseline recommendation approaches; section 6.2.2 includes the item-based baseline recommendation algorithms; and section 6.2.3 lists the diffusion kernels integrating with the RecWalk* model, as the graph-based recommendation algorithms.

6.2.1 Non-item Baseline Algorithms

As far as the non-item baseline algorithms were concerned, we selected two matrix factorization methods, *PureSVD* and *EigenRec*; two neural-based approaches, *multi-layer perceptron (MLP)* and *Generalized Matrix Factorization (GMF)*; a bipartite-graph based algorithm: (\mathbf{P}^3). Below are the non-item baseline recommendation algorithm:

- *PureSVD*: PureSVD: one of the classical Matrix Factorisation recommender methods, aiming to reduce the dimensionality of the matrix [CKT10]

- *EigenRec*: an extension of PureSVD that adds a scaling component for each item [CKT10]

- $\mathbf{P}^{(n=3)}$: a three-step random walk model on the user-item bipartite network, starting at a user node and ending with an item node where (n) must be an odd number [GPR+07]

- *Multi-Layer Perceptron (MLP)*: A multiple layers of the fully-connected neural networks as an implementation of the NCF framework as introduced in Chapter 2 [HLZ+17].

- *Generalized Matrix Factorization (GMF)*: A neural-based matrix factorization method that extracts the weights from the embedding layers of the users and items as the latent-factor vectors of the users and items, respectively [HLZ+17].

6.2.2 Item-based Baseline Algorithms

We choose *cosine (COS)* and *sparse linear method (SLIM)* as two item-based baseline recommendation approaches.

- *Cosine (COS)*: a neighbour-based similarity recommender method (ItemKNN) that measures the similarity of the row-wise or column-wise vectors from the user-item matrix [SKK+01].

- *SLIM*: a recommendation method which reconstructs the user-item matrix to build an item model by using ElasticNet Regression [NK11]

6.2.3 Kernel-based Algorithms

We select three diffusion kernels (PPR, LAP, and DR) as we already have introduced in chapter 4, and integrate each with the RecWalk* framework as *RecWalk*+PPR*, *RecWalk*+LAP*, and *RecWalk*+DR*, respectively.

6.2.4 Conclusion

In short, sections 6.2.1-6.2.3 indicate the algorithms used in our experiments. In the next section, we will detail our full experimental procedures, including the experimental results on eight public standard recommendation datasets.

6.3 EXPERIMENTS

This section details our experimental procedures on the standard recommendation datasets. In section 6.3.1, we first introduce the evaluation metrics, including the dataset partitioning strategy and the indicators of recommendation performance. Then, as discussed in section 6.3.2, we show the experimental results for the baseline recommendation methods and the RecWalk* model with diffusion kernels for each standard recommendation dataset. Finally, we compare the results of the RecWalk* model with diffusion kernels and the baseline recommendation methods on the standard recommendation datasets.

6.3.1 Evaluation

The standard *leave-one-out cross-validation* (LOOCV) [Cawo6] evaluation metric, widely used in traditional standard recommendation tasks, is known for its accuracy. This method randomly leaves out one sample from the dataset for testing while the rest data are used for training. The process is repeated for each sample in the dataset, ensuring that each sample is involved in training and testing. This strategy will give accurate recommendation results with low bias and low variance. However, it is essential to note that this method can be time-consuming and space-consuming, as it requires training and testing each sample in the dataset separately.

Therefore, we adopted the trivial version of the Leave-one-out cross-validation (LOOCV) [NK19] used by Karypis and Nikolopoulos to split each dataset into a train set (T_{train}) and a test set (T_{test}) in our experiments. The standard sampling strategy, Random Sampling - One of the items rated by a user was selected randomly and was used to partition the dataset. For a particular user, we consider their corresponding test item together with 1000 randomly sampled unrated items. These 1001 items are ranked according to their prediction scores generated by each recommender method. We repeated this process three times and recorded the average.

In our experiments, we use the hit rate (HR@N) as the evaluation metric for each algorithm. For a top-n recommendation task, the recommendation result to each user is a ranked list with the size of N. The HR gives the most objective evaluation of the recommendation list to check whether the tested item occurred in the recommended list or not. Our experiments tested the N from 1, 5, 10, 15, and 20, respectively, to each user. The overall HR score on the dataset, declared as the total number of Hits (#Hits) divided by the total number of users (#Users), is calculated as shown in eq 6.3.1.

$$\text{HR} = \frac{\text{\#Hits}}{\text{\#Users}} \quad (6.3.1)$$

In the next section, we will use the HR@N as the indicator to evaluate the performance of the recommendation algorithms on the eight public recommendation datasets (outlined in section 6.1), using the algorithms as introduced in section 6.2.

6.3.2 Results

This section presents our experimental results for the non-item baseline algorithms, the item baseline algorithms, and the RecWalk* model with diffusion kernels on the eight standard recommendation datasets. We used the HR@N as the indicator for each algorithm and applied 3-fold leave-one-out cross-validation on each dataset. Table 6.2 shows the experimental results (HR@10) where the rows represent the non-item baseline algorithms (PureSVD, EigenRec, MLP, GMF, and P^3), the item baseline algorithms (Base in Cosine and SLIM), and the RecWalk* model with diffusion kernels (LAP, PPR, DR). For instance, SLIM+RecWalk*+LAP represents using the SLIM item model to create the inter-item graph used within the RecWalk* model, and LAP is the Regularized Laplacian diffusion kernel. The columns represent eight recommendation datasets.

At the bottom of Table 6.2, we detailed the parameter settings for all algorithms.

Dataset	Non-item Baseline					Item Baseline \ RecWalk* \ Diffusion								P-Value
						Cosine				SLIM				
	SVD	EIG	MLP	GMF	P ³	Base	+PPR	+DR	+LAP	Base	+PPR	+DR	+LAP	
ML-1M	42.65	43.51	31.56	35.03	24.29	27.10	33.41	33.53	39.62	44.45	45.08	45.07	45.10	6.73×10^{-7}
YM	49.63	50.86	46.50	47.00	53.28	49.86	53.89	52.90	56.71	58.91	59.14	59.03	59.35	8.90×10^{-5}
AM-BY	13.01	14.60	11.39	10.17	13.55	10.41	14.67	14.96	15.14	15.35	15.82	16.06	16.10	5.77×10^{-5}
AM-AP	21.08	24.72	14.56	15.47	23.54	23.32	25.27	24.39	25.66	25.11	26.57	25.93	27.02	2.23×10^{-22}
AM-CP	18.32	21.88	14.55	10.62	23.40	16.78	24.41	25.71	26.14	25.59	26.41	26.18	26.64	1.53×10^{-7}
AM-HC	18.51	22.74	16.29	13.00	22.49	18.06	22.27	22.68	24.04	24.18	24.51	24.45	25.64	1.18×10^{-22}
BX	13.13	14.83	8.45	7.56	16.47	7.67	16.82	16.25	17.12	19.35	20.05	19.99	20.16	9.29×10^{-8}
SEM	58.16	63.61	52.51	50.21	67.71	61.59	68.63	66.62	69.13	71.62	71.79	71.92	72.27	1.69×10^{-7}

Parameters Settings: RecWalk*: $\theta \in \{0.005, 0.1, \dots, 0.9\}$. Diffusion Kernels: (+PPR/+DR/+LAP) $\beta \in \{0.1, \dots, 0.9\}$, DR (The step number) $k \in \{3, \dots, 10\}$. SVD/EIG: $f \in \{10, 20, \dots, 50\}$, $d \in \{0.1, \dots, 0.9\}$. P³: #Step = 3. Cosine: (The nearest neighbour) $k=200$. SLIM: $l1 - norm \in \{0.025, 0.2, 0.25\}$, $l2 - norm = 0.0001$. **Bold** number: The best performance of diffusion-based recommender methods

Table 6.2: Comparison of baseline (PureSVD:SVD, EigenRec:EIG and P³), item (Cosine and SLIM) and their RecWalk* framework with diffusion kernels (PPR, DR, and LAP). Evaluation metric: HR@10. Paired T-test (SLIM and SLIM+LAP) at the significance level (0.05)

As the results illustrated in Table 6.2, the diffusion-based RecWalk* model consistently outperforms the baseline item models – Cosine and SLIM – in terms of accuracy, and this difference is most noticeable in Cosine. Despite a much smaller improvement for SLIM, a paired ‘T-test’ was conducted for all datasets, and the difference was found to be significant at a p-value of the significance level of 0.05. (The p-value for LAP is shown in the ‘p-value’ column in the table). The LAP kernel performed better than the PPR and DR kernels in their overall accuracies. Therefore, the RecWalk* framework constructed using the SLIM baseline item model with the LAP kernel (SLIM+RecWalk*+LAP) consistently performed better than SLIM. Furthermore, the diffusion-based approaches performed better than non-item baseline algorithms, including PureSVD (SVD), EigenRec (EIG), MLP, GMF, and (P₃) in accuracy, except for the ML-1M dataset.

Additionally, to verify the correctness of the RecWalk* model, we compared the results (Normalized Discounted Cumulative Gain: NDCG@10 [NK19]) of the RecWalk* and the RecWalk model, which use Cosine as the baseline item model to examine whether the simplified weight initializing scheme of RecWalk* had any improvements. Table 6.3 shows that RecWalk* and RecWalk performed similarly on PPR and DR but that RecWalk* was better than RecWalk on LAP.

Model (Cosine)	+PPR	+DR	+LAP
RecWalk	20.33	19.82	17.84
RecWalk*	19.60	19.98	22.25

Table 6.3: Comparison of RecWalk* (+PPR/+DR/+LAP) and RecWalk (+PPR/+DR/+LAP) on Cosine (ML-1M). Evaluation Metric: NDCG@10

Next, we show the top-n recommendation accuracy (HR@N) of the non-item baseline algorithms and item/diffusion-based algorithms for each dataset in Figures 6.1-6.3 below, where N is chosen from {1,5,10,15, and 20}.

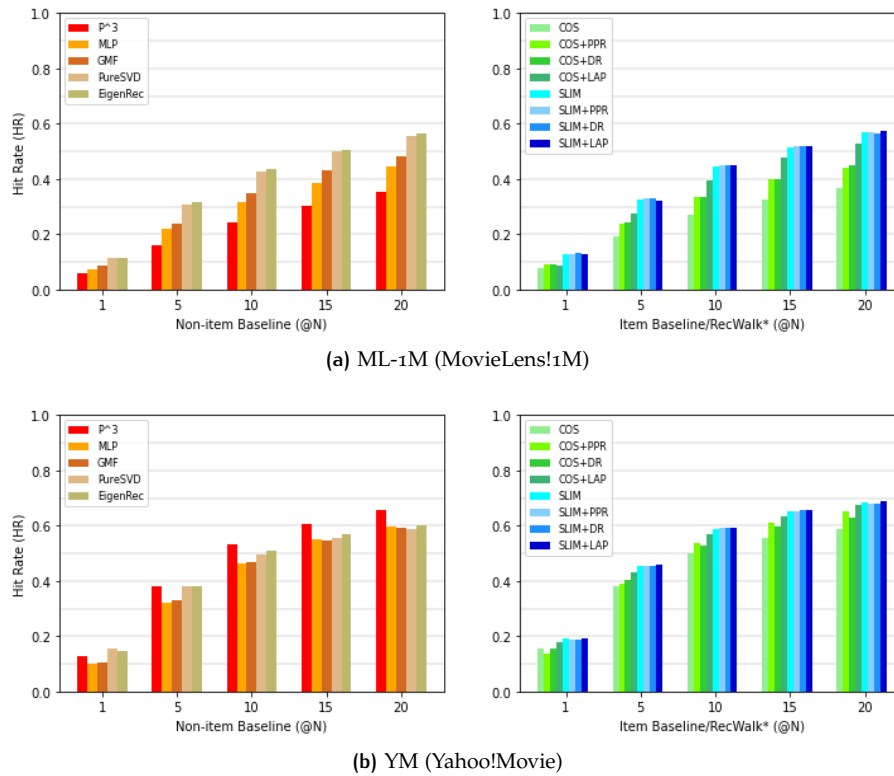
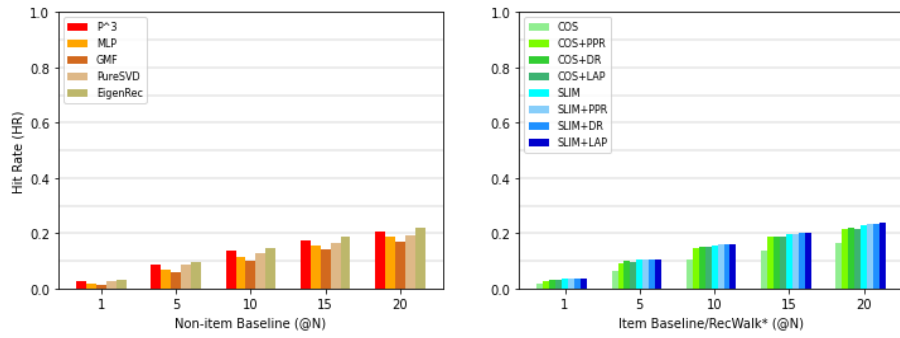


Figure 6.1: Experimental results (ML-1M and YM)

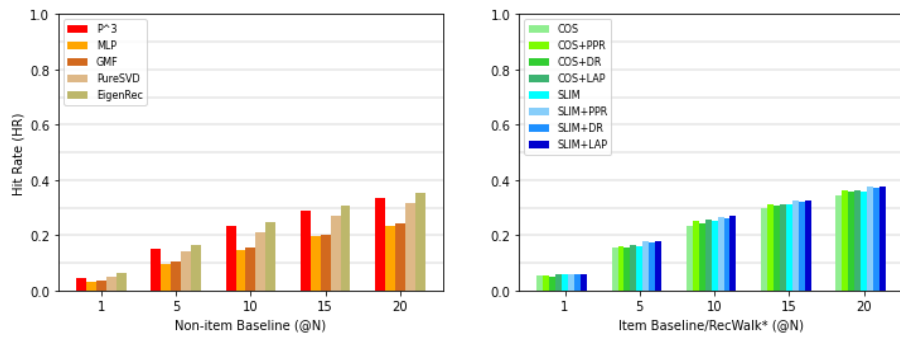
Figures 6.1 (a) and (b) show the complete experimental results for two movie-based datasets: ML-1M and YM. In each plot, the left graph represents the results of all non-item baseline algorithms, including (\mathbf{P}^3 , PureSVD, EigenRec, MLP, and GMF); the right graph shows the results of the item-baseline models (Cosine: COS and SLIM) and their RecWalk* framework with diffusion kernels (+PPR/+DR/+LAP), respectively. The x-axis represents (N), the number of recommended items, and the y-axis represents the average hit rate (HR@N). For the ML-1M dataset, we found that EigenRec was the best baseline recommendation algorithm, and RecWalk*+COS+LAP improved significantly than the baseline COS model. For the YM dataset, we found that \mathbf{P}^3 was the best baseline algorithm, and RecWalk*+COS+LAP improved the accuracy of the baseline COS model but not as significantly as the ML-1M dataset.

Figures 6.2 (a)-(d) show the complete experimental results for the Amazon-series datasets, including AM-BY, AM-CP, AM-AP, and AM-HC. Compared with the movie-based datasets shown in Figure 6.1, the overall accuracy of Amazon-series datasets is lower than that of movie-series datasets. For the Amazon-series datasets, \mathbf{P}^3 and EigenRec were the best non-item baseline recommendation algorithms, and the RecWalk*+diffusion kernels improved the performance of the baseline item models but not significantly as the movie-series datasets.

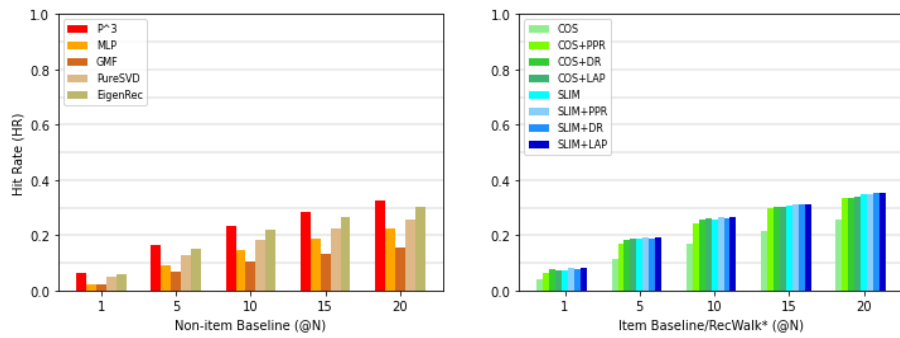
Figures 6.3 (a)-(b) show the full experimental results for the BX and SEM datasets. For these two datasets, \mathbf{P}^3 was the best non-item baseline recommendation algorithm. The diffusion-based RecWalk* algorithms improved the performance of the baseline item models and were better than the baseline non-item models.



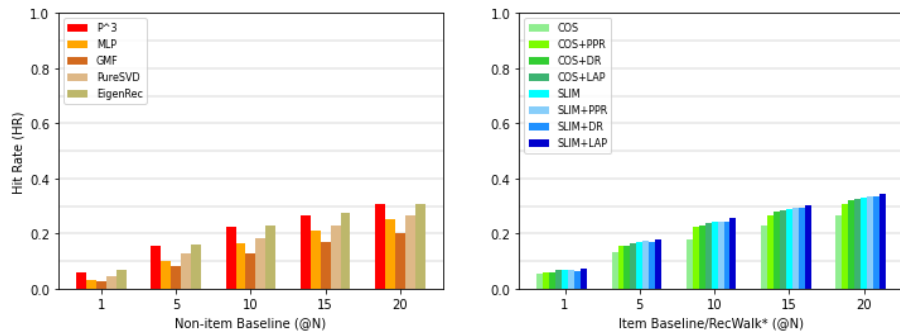
(a) AM-BY (Amazon-Baby)



(b) AM-AP (Amazon-Apps for Android)



(c) AM-CP (Amazon-Cell Phones and Accessories)



(d) AM-HC (Amazon Health and Care)

Figure 6.2: Experimental results (AM-BY,AM-CP,AM-AP,and AM-HC)

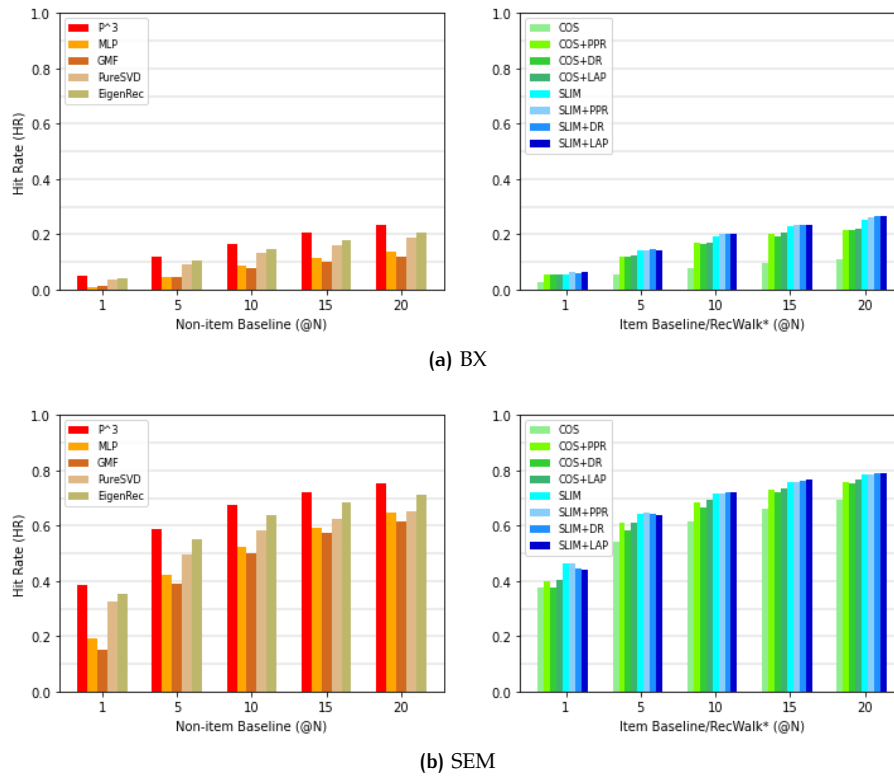


Figure 6.3: Experimental results (BX and SEM))

6.4 CONCLUSION

Overall, this chapter summarizes the evaluation of kernel-based recommendation methods, including the standard recommendation datasets, algorithms, evaluation metrics, and experimental results.

In section 6.1, we first introduced eight public standard recommendation datasets. We applied the data preprocessing techniques on some datasets to filter out those sparse users with less than two rated items and sparse items rated by less than two users. Then, we outlined our algorithms, including baseline algorithms (item-based and non-item-based) and graph-based algorithms (RecWalk* model + diffusion kernel) in section 6.2. Afterwards, section 6.3 sets up the evaluation benchmark. In section 6.3.1, we adopted the LOOCV as the dataset partitioning metric to divide the original dataset into train and test sets and used the hit rate as the primary evaluation indicator. In section 6.3.2, we conducted experiments on these eight public recommendation datasets with the baseline algorithms and the RecWalk*+diffusion kernels and produced the data tables and graphs to present their results.

Based on our experimental results, we concluded that the RecWalk*+diffusion kernels performed consistently better than other baseline algorithms, and RecWalk*+LAP consistently performed better than others. However, such a method only mines the information within the user-item interaction data but ignores the external semantic representations about users or items. Interestingly, these datasets exhibited the recommendation accuracy at different levels. For example, the Amazon-series datasets are lower than others; the Steam Video Game dataset performed signifi-

cantly better than others. This phenomenon may be due to the rating distribution of the datasets.

In the next chapter, we will discuss how to mine more valuable information about items from a knowledge graph and how to link the standard recommendation graph to a knowledge graph to contribute to the recommendation result.

7

EXTENDING KERNEL-BASED RECOMMENDATIONS WITH A KNOWLEDGE GRAPH

7.1 MOTIVATION

In Chapters 5 and 6, we have introduced the non-item baseline algorithms, the item baseline algorithms, and the RecWalk* model with diffusion kernels and conducted experiments on eight well-known public standard recommendation datasets from different domains. From their experimental results, we concluded that the graph-based RecWalk* model with diffusion kernels outperformed the non-item and item baseline algorithms. However, we only used the user-item interaction information provided by the standard recommendation dataset but did not consider any additional content about the items or users. We have used the user-item bipartite graph and the inter-item graph. However, now we would like to add another graph to the dataset, a portion of the knowledge graph, which contains the items' semantic information.

In this chapter, section 7.2 first introduces knowledge graph, which includes the additional semantic information about items in the standard recommendation datasets. Then, we will move on to some standard recommendation datasets. Moreover, we are going to show how we can link the items in the recommender systems to the subset of the knowledge graph, and how we intend to use the subset of the knowledge graph to extend the current standard recommendation graph.

Afterwards, we will introduce the knowledge graph extraction technique to extract the knowledge subgraph for each item via the deterministic graph traversals, as discussed in section 7.4. Next, we will introduce the knowledge graph filtering technique to remove the irrelevant relations to get an accurate and concise filtered knowledge subgraph, as introduced in section 7.5.

7.2 KNOWLEDGE GRAPHS

We have mentioned the knowledge graph in Chapter 2 to extend the semantic representations of items used in the content-based recommendation approaches. In this chapter, we will explain the concept of knowledge graphs in detail. As a type of linked data structure, a knowledge graph is a knowledge base that describes or interprets entities and relationships between them [FŞA+20]. An extensive knowledge graph can be a warehouse of millions of things or facts (E.g., DBpedia is the most extensive knowledge graph consisting of 850 million facts in the version of Sep 2021). Each fact is a triplet [FŞA+20] consisting of three parts: a head entity, a tail entity, and

Entity Name	Entity URI
Different for Girls	http://dbpedia.org/resource/Different_for_Girls

Table 7.1: Entity URI representation

a relation that connects the head entity and the tail entity. All entities from the knowledge are made up of the entity set (E), and all relations from the knowledge graph are made up of the relation set (R). E.q (7.2.1) defines a knowledge graph (KG) as a collection of triplets (e_1, r, e_2) where (e_1) and (e_2) represent the head and tail entities, respectively, from the entity set (E), and (r) is a type of relation from the relation set (R).

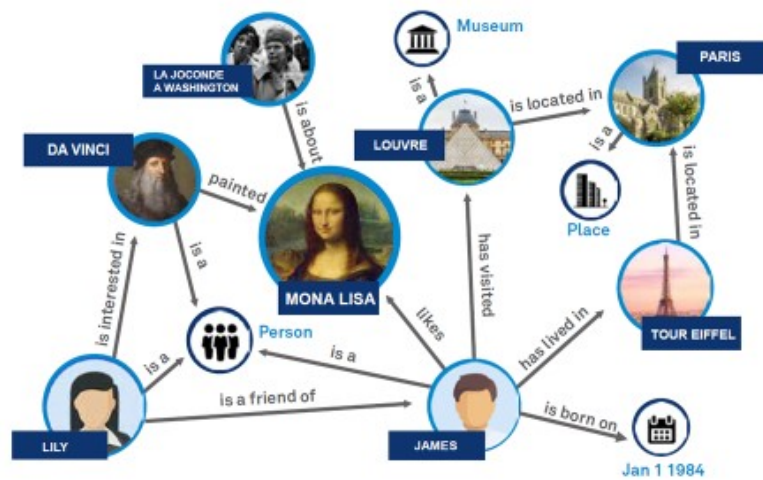


Figure 7.1: Knowledge graph visualization example (DBpedia) [Cha21]

$$KG = \{(e_1, r, e_2) | e_1, e_2 \in E, r \in R\} \quad (7.2.1)$$

We used the DBpedia snapshot (Figure 7.1) as an example to illustrate a triplet of the graph. For instance, the triplet $(\text{'JAMES'}, \text{'has lived in'}, \text{TOUR EIFFEL})$ is a triplet. In this example, the person, 'JAMES', and the place, 'TOUR EIFFEL', are the head entity and tail entity, respectively, and 'has lived in' is a relation.

In addition, each entity or relation in the knowledge graph is represented by a Universal Resource Identifier (URI) [BFM04]. People commonly use a webpage to store the resource information of an entity or a relation. For instance, Table 7.1 shows the URI representation of the movie 'Different For Girls' from the DBpedia knowledge graph.

7.3 DATA

In Chapters 5 and 6, we only used the user-item interactions of the standard recommendation dataset in the collaborative filtering task. However, we did not consider the semantic information about users or items. The semantic information about users or items commonly refers to their semantic properties stored in the nodes of the knowledge graph. We link the items in the standard recommendation dataset onto their corresponding entity URIs in the knowledge graph using the *entity linking technique (EL)* [HRN+13]. In section 7.3.1, we introduce three standard recommendation datasets and then continue the entity linking technique in section 7.3.2.

7.3.1 Standard Recommendation Datasets & Results

In Chapter 6, we introduced eight standard recommendation datasets, but most only provided the user-item interaction data. This chapter will outline another three standard recommendation datasets providing the items' description profiles. The items' description profile contains the item's basic information, like name, which helps us locate their entity URI in the knowledge graph. These three datasets include *ML-300K*, a filtered version of *ML-1M* by removing the hub users or items; *Facebook-Music (FB-MS)* [NOT+16], consisting of the ratings users give to the artist on Facebook; and *LibraryThing (LT)* [NOT+16] including the rating users give to the books in a social cataloguing web application. The statistics of these three datasets are shown in Table 7.2 below. For each dataset, we present the number of users (#Users), the number of items (#Items), the number of interactions (#Links) between users and items, and the density (Density) of the dataset. For each dataset, we ensure that each user has at least eight rated items, and each item is shared at least by eleven users.

Dateset	#Users	#Items	#Links	Density
ML-300K	4300	3035	305,392	2.34%
FB-MS	5735	4379	242,002	0.96%
LT	4544	3680	151,315	0.90%

Table 7.2: Statistics of three standard recommendation datasets

7.3.2 Entity Linking

As we mentioned in the introduction of this Chapter, we can connect an item in the standard recommendation dataset to the entity node of the knowledge graph via items' URIs. *Entity Linking* [HRN+13] is a standard way of linking an item node in a standard recommendation dataset using the description profile to an entity node in a knowledge graph. For the three standard recommendation datasets outlined in section 7.3.1, Di et al. [DMO+12] has provided the processed URI mapping profile for items using the standard entity linking technique. Figures 7.2 (a) – (c) show the examples of the URI mapping profile for the *ML-300K*, *FB-MS*, and *LT* datasets, respectively. As shown in Figure 7.2 below, the first column represents the 'itemID' in the stan-

3280	Baby, The (1973)	http://dbpedia.org/resource/The_Baby_(film)
3282	Different for Girls (1996)	http://dbpedia.org/resource/Different_for_Girls
3284	They Might Be Giants (1971)	http://dbpedia.org/resource/They_Might_Be_Giants_(film)
3283	Minnie and Moskowitz (1971)	http://dbpedia.org/resource/Minnie_and_Moskowitz
(a) ML-300K		
2793		http://dbpedia.org/resource/The_Claudi_Journals
1462		http://dbpedia.org/resource/Rising_Sun_(novel)
2792		http://dbpedia.org/resource/Full_Tilt_(novel)
1461		http://dbpedia.org/resource/Shadow_Puppets
(b) FB-MS		
391054	The Way to Dusty Death	http://dbpedia.org/resource/The_Way_to_Dusty_Death
173676	Flyte	http://dbpedia.org/resource/Flyte
75592	The Sibley Guide to Birds	http://dbpedia.org/resource/The_Sibley_Guide_to_Birds
173670	The Lightning Thief	http://dbpedia.org/resource/The_Lightning_Thief
(c) LT		

Figure 7.2: Items' URIs mapping profile

standard recommendation dataset, and the last column represents the corresponding item URI. The ML-300K and the LT dataset also include the items' titles. We created a dictionary (D) for each standard recommendation dataset to map each 'itemID' onto its corresponding URI.

As the mapping profile contains the mapping information for each item paired with its identifier and entity URI, we are ready to find the candidate relations to extract the knowledge subgraph starting with the item URI.

7.4 KNOWLEDGE GRAPH EXTRACTION

This section will introduce the *Knowledge Graph Extraction (KGE)* [WMC+18] technique for obtaining a local knowledge subgraph for each item as its additional semantic representation.

We use *DBpedia* as the source knowledge graph [ABK+07]. *DBpedia* is a graph database comprising 850 millions of triplets with the version of Sep 2021 [Hol21]. In a recommendation task, we only need to find the parts related to the items in the standard recommendation dataset. Thus, we define a rule to extract the *DBpedia* knowledge graph to obtain the local knowledge subgraph of each item in the standard recommendation dataset. We start with the URI for each item and select a set of ontology-based relations as the candidate relations. An *ontology* is a fundamental concept from the information science domain that abstracts the standard properties for some things. For instance, 'Film' is an ontology that includes properties such as 'author', 'director', and 'stars'. The *DBpedia ontology* is generated from the manually created specifications in the *DBpedia Mappings Wiki* [LJ+15]. The *DBpedia Mappings Wiki* provides information about all properties related to the target ontology and allows users to enquire about the properties related to the target ontology. Our work mainly emphasizes the English version. Figure 7.3 above shows some properties related to the Film ontology found in the *DBpedia Mappings Wiki*. In this table, the 'Name' column represents the name of a property; the 'Label' column represents the label of a property; the 'Domain' column gives the superclass of a property; and the 'Range' column

Properties on Film:

Name	Label	Domain	Range
afdbld (edit)	afdb id	Film	<i>xsd:string</i>
allcinemald (edit)	allcinema id	Film	<i>xsd:string</i>
alternativeTitle (edit)	alternative title	Work	<i>rdf:langString</i>
amgid (edit)	amgld	Film	<i>xsd:string</i>
author (edit)	author	Work	Person
basedOn (edit)	based on	Work	Work
bgafldd (edit)	bgafld id	Film	<i>xsd:string</i>
bibo:pages (edit)	pages	Work	<i>xsd:string</i>
chiefEditor (edit)	chief editor	Work	Person
cinematography (edit)	cinematography	Film	Person

Figure 7.3: Properties of the 'Film' Ontology in DBpedia

gives the concrete class of a property.

In our work, we aim to explore the properties which give the most accurate description of the items in the standard recommendation datasets. Firstly, we choose some properties from the DBpedia Mappings Wiki search results as candidate relations. For example, the 'author' property provides the authors' information about the film, and the 'chief editor' property lets us know the information about the chief editors of the film. However, some property like 'amgid' or 'bgafldd' refers to the administration domain but not relative to our work. Therefore, we only selected those properties (e.g., 'author' and 'chiefEditor') which are relevant to the Film domain as the candidate relations, and filtered out those properties (e.g., 'bgafldd' and 'amgid') which belong to the administration domain.

Section 7.4.1 introduces a knowledge graph extraction approach. We will apply this approach to extract the knowledge subgraphs from DBpedia for the items in the three standard recommendation datasets.

7.4.1 DFSPARQL

In this section, we propose a method, 'DFSPARQL', that recursively extracts the knowledge subgraph from a source knowledge graph. The DFSPARQL stands for using SPARQL to extract the knowledge subgraph from the source knowledge graph via DFS (a deterministic graph traversal technique as we already introduced in Chapter 2).

SPARQL is a structural querying language used in a graph database. The SPARQL fetches a record from the graph database as a triplet, including the URIs of entities or relations. The DFSPARQL method uses DFS, a deterministic graph traversal, as mentioned in Chapter 3, and SPARQL to extract the knowledge subgraph of an item in the standard recommendation dataset.

The screenshot shows a SPARQL query interface. The query is as follows:

```

1 PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
2 PREFIX rdfs: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
3 PREFIX dbr: <http://dbpedia.org/resource/>
4 PREFIX foaf: <http://xmlns.com/foaf/0.1/>
5 PREFIX dct: <http://purl.org/dc/terms/>
6 PREFIX dbr-owl: <http://dbpedia.org/ontology/>
7 PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
8
9 SELECT * WHERE {
10   <http://dbpedia.org/resource/Stealing_Beauty> ?rel ?res.
11 }
12

```

The results are displayed in a table view with the following columns:

rel	res
267 <http://dbpedia.org/property/country>	==@en
270 <http://dbpedia.org/property/distributor>	==@en
277 <http://dbpedia.org/property/language>	==@en
283 <http://dbpedia.org/property/productionCompanies>	==@en
287 <http://dbpedia.org/property/runtime>	==@en
291 <http://dbpedia.org/property/starring>	==@en
107 dbr-owl:wikiPageRevisionID	*1115508113**<http://www.w3.org/2001/XMLSchema#integer>
319 dbr-owl:Work/runtime	*113.0**<http://dbpedia.org/datatype/minute>
320 dbr-owl:Work/runtime	*119.0**<http://dbpedia.org/datatype/minute>
321 dbr-owl:wikiPageLength	*16869**<http://www.w3.org/2001/XMLSchema#nonNegativeInteger>

Figure 7.4: Querying result of the SPARQL

We used *Triply* [Tri24], an open-source software, to query and manage the knowledge graph. Figure 7.4 above shows an example of the SPARQL querying result using the Triply platform. In the 'Query' section, we aim to search for the relations and properties related to the movie, 'Stealing Beauty'. We set the URI of this movie in the subjective clause of this query and create two variables, 'rel' and 'res', to record the querying results for the relations and properties, respectively. The 'Table' section shows all querying results given by their URIs.

However, we only want to fetch the properties most semantically relevant to the item from the standard recommendation database. Therefore, we can use the candidate relations chosen from the DBpedia Mappings Wiki search results as the querying relations in the SPARQL clause.

Algorithm 7.1: SimpleSubKGExtraction

Input: an item ($c \in I$), item-URI mapping dictionary (D), candidate relations set ($R_s \subseteq R$), DBpedia knowledge graph (G)

Output: a knowledge subgraph G_s

```

1 Assign uri : uri  $\leftarrow$  D[c]
2 Assign q : q  $\leftarrow$  qpre
3 Initialize  $G_s$  :  $G_s \leftarrow \emptyset$ 
4 ind  $\leftarrow$  0
5 foreach r  $\in$   $R_s$  do
6   if ind == 0 then
7     | Assign  $q_s$  :  $q_s \leftarrow \{< uri > \quad r \quad ?obj.\}$ 
8     end
9   else
10    | Update  $q_s$  :  $q_s \leftarrow q_s \cup \{< uri > \quad r \quad ?obj.\}$ 
11    end
12    ind  $\leftarrow$  ind + 1
13    Append q : q  $\leftarrow$  q +  $q_s$ 
14 end
15 Collect  $G_s$  :  $G_s \leftarrow$  SPARQL(q)

```

Algorithm 7.1 presents a simple knowledge subgraph extraction procedure for an item to extract all one-hop neighbours connected to the item on the knowledge graphs using SPARQL. We first declare some terms in this algorithm. We create a dictionary (D) that maps each item in the standard recommendation dataset onto their entity URI in the knowledge graph, a full-relation set (R) representing all relations existing in the DBpedia knowledge graph (G), and a sub-relation set (R_s) representing the candidate relations for querying, and a knowledge subgraph (G_s) as the output.

Algorithm 7.1 accepts four inputs, including an item (c), an item-URI mapping dictionary (D), a set of candidate relations (R_s), and the DBpedia knowledge graph (G). This algorithm outputs a knowledge subgraph (G_s) to the input item (c). The first line declares a variable (uri) to store the URI of the input item by looking up the dictionary (D). Line 2 declares a variable (q) to store the complete SPARQL query, and we initialize this variable as the common prefix of the URIs (q_{pre}). Line 3 declares the output knowledge subgraph (G_s) and initializes it as empty. Line 4 declares a variable 'ind' to indicate the number of candidate relations. If the 'ind' is equivalent to zero (Line 6), we will create a new SPARQL query clause and store it into the variable (q_s) as shown in Line 7; otherwise, we will repeatedly append the query to the variable (q_s) for each candidate relation as shown in Line 10. In lines 7 and 10, we use a bracket to represent a single SPARQL query clause. A single SPARQL query clause consists of three components: the querying entity $<uri>$, the querying relationship (r), and the querying result variable ($?obj$). Line 13 executes the query on the knowledge graph using SPARQL and returns the knowledge subgraph (G_s) to the item (c).

In short, this algorithm starts with the item URI and extracts all one-hop neighbour nodes filtered by the candidate relations in the knowledge graph. However, the one-hop neighbour nodes are insufficient to provide accurate semantic information about the item, as we want to look deeply at the knowledge graph for extracting the two-hop or more-hop neighbour nodes. Therefore, we provide the DFSPARQL approach, as shown in Algorithm 7.2.

Algorithm 7.2: DFSPARQL

Input: an item ($c \in I$), item-URI mapping dictionary (D), candidate relations set ($R_s \subseteq R$), DBpedia knowledge graph (G), traversal depth (d)

Output: a knowledge subgraph G_s

```

1 Initialize  $G_s : G_s \leftarrow \emptyset$ 
2  $G'_s \leftarrow \text{SimpleSubKGExtraction}(c, D, R_s, G)$ 
3  $G_s \leftarrow G_s \cup G'_s$ 
4 if  $d == 1$  then
5   | Collect  $G_s$ 
6 end
7 else
8   | foreach  $e \in G'_s$  do
9     |  $G''_s \leftarrow \text{DFSPARQL}(e, D, R_s, G, d - 1)$ 
10    |  $G_s \leftarrow G_s \cup G''_s$ 
11   | end
12 end

```

Algorithm 7.2 recursively explores all n -hop neighbours connected to the item on the knowledge graph using the Depth-first Search. Compared with the ‘SimpleSubKGExtraction’ algorithm discussed in Algorithm 7.1, we add the traversal depth (d). Line 1 initializes an empty graph (G_s) to store the output knowledge subgraph. Line 2 first invokes Algorithm 7.1 to obtain all one-hop neighbours using all candidate relations and stores the result as a temporary graph (G'_s). Line 3 concatenates all one-hop neighbours onto the graph (G_s). Lines 4-10 implement the recursive Depth-first Search graph traversal to extract the d -hops neighbours. Iteratively, for each URI obtained in the result (G'_s) at the depth ($i - 1$) ($1 \leq i \leq d$), we extract all one-hop neighbours (G''_s) at the depth (i) until the traversal depth reaches the target depth (d), as shown in lines 8-10.

We demonstrate the unfiltered knowledge subgraph construction process for the three standard recommendation datasets, including ML-300K, FB-MS, and LT, as shown in Appendix: Section A. We choose the top five highly-occurring relations for each standard recommendation dataset as the candidate relations to create the unfiltered knowledge subgraph. However, these relations are only manually analyzed and extracted. Some of the relations may make noise in the knowledge subgraph. Therefore, it is necessary to filter the knowledge subgraph to obtain a concise and accurate version.

7.5 KNOWLEDGE GRAPH FILTERING

In section 7.4, we used the candidate relations set to extract an unfiltered knowledge subgraph for an item in the standard recommendation dataset. Although those candidate relations are the semantic properties of the items, some may bring the noises onto the knowledge graph. Therefore, we choose the subsets of relations that clearly and accurately reveal the relations between a pair of items. We first select the number of pairs of positively related entities and pairs of negatively related entities. Then, we compare the similarities between the pairs of positively

related entities and the pairs of negatively related entities using the embedding-based approach. Based on their similarity scores using the embedding-based approach, we choose the subsets of relations from the primary candidate relations set to build the filtered knowledge graph for each item.

In this section, we focus on the knowledge graph filtering. We aim to filter the knowledge subgraphs we constructed in section 7.4 to accurately reflect the semantic relatedness between items. To estimate the semantic relatedness for a pair of items, section 7.5.1 first manually categorizes each pair into different groups based on the number of shared labels or properties in the knowledge graph. Then, section 7.5.2 introduces a scoring system to accurately calculate each item pair's semantic relatedness score. We demonstrate the knowledge subgraphs filtering procedure for three unfiltered knowledge subgraphs as presented in section A in the appendix, and the filtered knowledge subgraphs are shown in section B in the appendix.

7.5.1 Shared Categorical Labels

To estimate the semantic relatedness for a pair of items, we first divide all items into pairs based on the number of their shared labels or properties in the knowledge graph. As the '*dc-terms:subject*' relation gives the categorical information of an item, we can calculate the number of the shared categorical labels between a pair of items and use this number to roughly define their semantic relatedness. However, the number of shared labels varies depending on the datasets. Therefore, we proposed a scoring system, *Ratio of the Shared Categorical Labels (RSCL)*, as shown in Eq 7.5.1 below. In this equation, the RSCL score between items is defined as dividing the number (#SCL) of shared labels between the pair of items and the maximum number $\text{Max}(\#SCL)$ of the shared labels among all pairs of items.

$$\text{RSCL} = \frac{\#SCL}{\text{Max}(\#SCL)} \quad (7.5.1)$$

RSCL	Relatedness
$\geq 75\%$	Strongly related
$\geq 50\%$ and $< 75\%$	Mediumly related
$< 50\%$	Weakly related
0	Unrelated

Table 7.3: RSCL score and relatedness checklist

Table 7.3 defines the degree of semantic relatedness for a pair of items using the RSCL score. For a pair of items, we consider they are 'Strongly related' when their RSCL score is more excellent than 75%; 'Mediumly related' when their RSCL score is between 50% and 75%; 'Weakly related' when their RSCL score is less than 50% but above than zero; and 'Unrelated' when they have no shared categorical labels. In short, two items are considered strongly related if shared with more categorical labels; otherwise, they are weakly related or unrelated.

7.5.2 Item Embeddings

Section 7.5.1 introduced the RSCL scoring scheme, which roughly measures the semantic relatedness of a pair of items and classifies them into different groups. However, such a scheme only roughly measures the semantic relatedness of a pair of items based on their shared categorical information. We need to figure out which relations are good or bad. We aim to find the relations that accurately reflect the semantic relatedness of a pair of items. Thus, we create items' embedding vectors as the semantic feature vectors so that we can calculate their semantic similarity scores using their embedding vectors.

We used TransE, a knowledge graph embedding technique argued by Bordes et al. [BUG+13], to translate the entities and relations of the knowledge graph onto embedding vectors. The TransE embedding approach accepts two inputs: a learning rate (r) and an embedding size (K). We tested the learning rate from 0.001, 0.002, 0.005, 0.01, 0.05, 0.1 and the embedding size from 10, 50, 100, 200.

Afterwards, we use the Cosine similarity to calculate the semantic similarity scores between items using their embedding vectors. As we mentioned, a good relation should give high similarity scores for a pair of strongly related items but low similarity scores for unrelated items. Therefore, we aim to find the relations which accurately interpret the semantic similarities between items.

We apply the knowledge graph embedding approach to the unfiltered knowledge subgraph introduced in Appendix: section A and obtain the filtered knowledge subgraph for each standard recommendation dataset in Appendix: section B.

7.5.3 Embedded-item Model

This section mainly discusses constructing the embedded-item model to represent the semantic similarity between a pair of items. Chapter 3 introduced the item model constructed from the user-item matrix using the collaborative filtering method. An item model can be represented as an inter-item matrix or an inter-item graph. Similarly, an embedded-item model has two representations: an embedded-item matrix and an embedded-item graph.

We have used TransE to transform the items' knowledge subgraph onto entity embedding vectors. Using the cosine similarity measure, we can calculate the similarity of a pair of items' embedding vectors as the semantic similarity scores for the pair of items. However, the embedding features are usually high-dimensional and complicated. We aim to apply the feature engineering technique to simplify the original embedding features to obtain precise and interpretable features.

Therefore, we apply K-Means++ [BMV+12], a clustering algorithm on the items' TransE embedding vectors, to group similar items onto clusters and calculate the centroid vector of each cluster. For instance, in a movie database, a cluster may refer to a genre like 'Romantic'; in a book database, a cluster may refer to a theme like a 'novel'. Then, we transfer each item's TransE

embedding vector into their counter-similarity embedding vector. *Counter-similarity*, argued by Byeong in 2006 [KLP+06], gives a likelihood of an item belonging to a cluster class using the Euclidean distance between the TransE item’s embedding vector and the centroid’s embedding vector.

$$\text{CounterSimilarity}(i, c_k) = 1 - \frac{D_{\text{EU}}(i, c_k)}{D_{\text{EU}}^{\text{Max}}(i, c_j)} \quad (7.5.2)$$

Eq 7.5.2 defines the counter-similarity between the item (i) and the cluster (c_k). $D_{\text{EU}}(i, c_k)$ represents the Euclidean distance between the TransE embedding vector of the item (i) and the centroid TransE embedding vector for a cluster, and $D_{\text{EU}}^{\text{Max}}(i, c_j)$ represents the maximum Euclidean distance of the item (i) among all cluster centroids. The counter-similarity reduces the dimensionality of the embedding vectors and gives more interpretable results. We collect the counter-similarity score of an item and send it to each cluster together as the counter-similarity embedding vector for the item. Each entry within the counter-similarity embedding vector represents the likelihood of an item belonging to a cluster. Afterwards, we used cosine to calculate the ground-truth similarity scores between items using their counter-similarity embedding vectors. We only kept the item pairs whose similarity scores were greater than 0.5.

The ‘*elbow method*’, argued by Thorndike in 1953 [Tho53], is an approach to determining the number of clusters in a dataset. The method aims to find a cutoff point as the ‘knee of a curve’ from the (squared sum of error) SSE Loss versus the cluster number curve. With the increasing number of clusters, the SSE Loss declines dramatically until reaching the cutoff point. After the cutoff point, the SSE Loss declines steadily. Therefore, the number of clusters at the cutoff point is considered the best cluster number. We adopted the ‘elbow method’ to find the optimal cluster number for the TransE embedding vectors.

At the end of this section, we demonstrate the cluster number optimization procedure for the items’ TransE embedding vectors used in three knowledge graphs (ML-300K-KG, FB-MS-KG, and LT-KG) in Figure 7.5 below. We apply the SSE loss function to the KMeans++ algorithm. Figure 7.5 contains three subplots representing the cluster number optimization of the KMeans++ algorithm. In each plot, the x-axis represents the candidate cluster number (C) from 1 to 100, and the y-axis represents the SSE loss at each cluster number.

The cutoff point of the ML-300K-KG, FB-MS-KG, and LT-KG is 30, 20, and 20, respectively. Therefore, we resize the dimensionality of the TransE embedding vectors into 30, 20, and 20 for the entities of three knowledge graphs, respectively. Afterwards, we create the new embedding vectors using the counter-similarity. Finally, we build the embedded-item model using the counter-similarity embedding vectors and filter out those links whose counter-similarity scores were less than 0.5.

Once we obtain the embedded-item model, we will propose a data augmentation method to enrich the user-item interactions in the standard recommendation dataset. The following section

will discuss how we generate a small number of non-existent links using the embedded-item model.

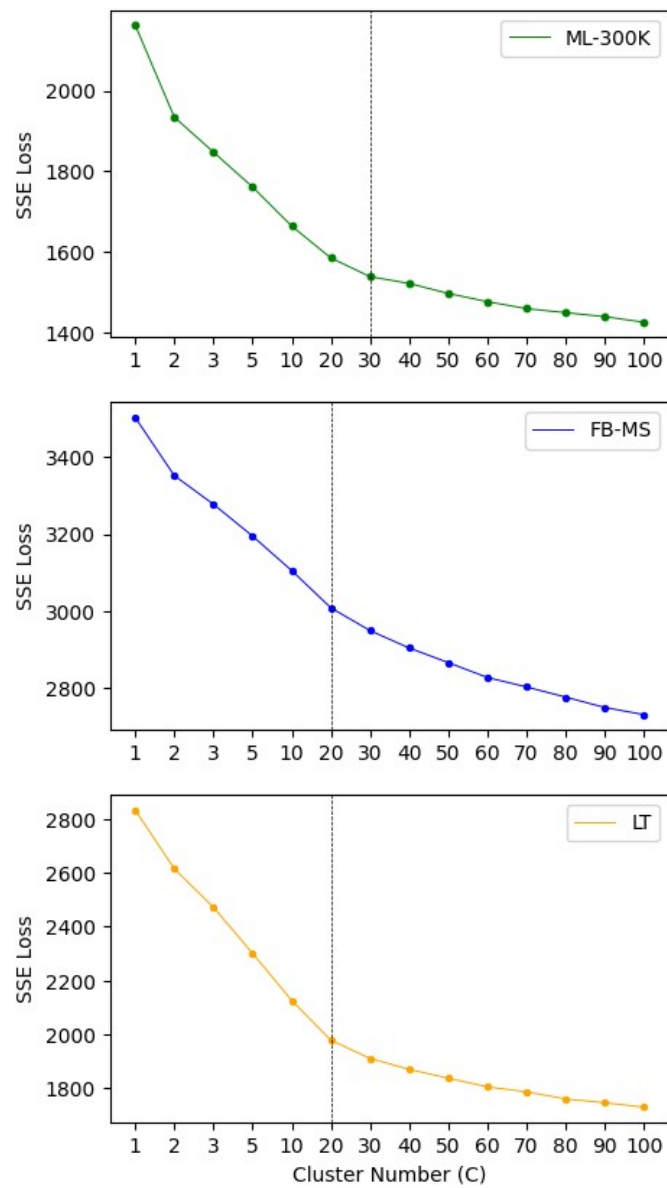


Figure 7.5: Cluster number optimization: entity embedding vectors for three datasets: (ML-300K, FB-MS, and LT)

7.6 KGRECWALK* MODEL

As we have obtained the filtered knowledge subgraph representing the semantic information about the items, we aim to modify the current RecWalk* framework with the filtered knowledge subgraph. One solution is to merge the user-item combination graph and the filtered knowledge subgraph as a graph by the item nodes. As discussed in Section 5, a random surfer can traverse between the user and item nodes randomly on the user-item combination graph with a biased

coin-tossing probability. Inspired by that, we can change the coin-tossing probability mechanism to fit these three subgraphs: the user-item bipartite graph, the inter-item interaction graph, and the filtered knowledge subgraph. Therefore, we propose the new framework, KGRecWalk*. Compared with the biased coin-tossing with two outcomes ('Head' and 'Tail'), as used in the RecWalk* framework in Chapter 5, we provided three outcomes ('Head', 'Medium', and 'Tail') for a biased coin-tossing. We use (θ_1) to denote the probability of a coin that yields on the 'Head', (θ_2) to denote the probability of a coin that yields on the 'Medium', and (θ_3) to represent the probability of a coin that yields on the 'Tail', respectively.

Figure 7.6 below sketches the KGRecWalk* framework. In this figure, the green and orange nodes represent the user and item nodes, respectively, the same as those used in the RecWalk* framework. And we use the purple nodes to represent the entity nodes. Eq 7.6.1 demonstrates a block matrix where the first and second columns represent all users and item nodes, respectively, in the user-item bipartite graph, and the last row represents entity nodes in the knowledge graph. The purple nodes denote the property nodes (for example, 'C1', 'C2', and 'C3') in the knowledge graph. Eq 7.6.2 defines the transition probability matrix (\mathbf{P}) of the KGRecWalk* framework. The transition probability matrix (\mathbf{P}) consists of three main parts: the user-item transition probability matrix, the inter-item transition probability matrix, and the knowledge graph transition probability matrix. The declaration of the user-item transition probability matrix and the inter-item transition probability matrix in the KGRecWalk* framework is identical to that in the RecWalk* framework. For the knowledge graph component, we use (\mathbf{W}_{ig}) to represent the transition probability matrix between the item nodes and the property nodes, and (\mathbf{W}_{kg}) to represent the transition probability matrix between the property nodes. In addition, the coin-tossing probability that yields on all conditions ('Head', 'Medium', 'Tail') is summing up to 1, as shown in Eq.7.6.3 below.

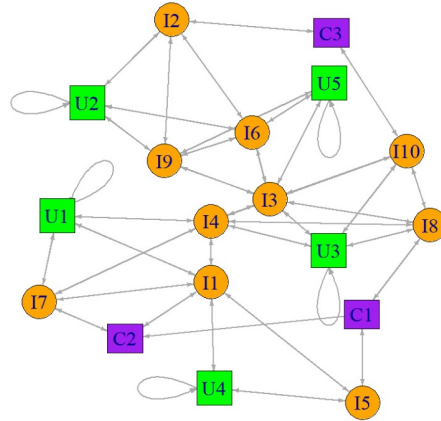


Figure 7.6: KGRecWalk* Structure

$$\mathbf{H}_{\text{KGRecWalk}^*} \leftarrow \begin{bmatrix} 0 & \mathbf{R}_x & 0 \\ \mathbf{R}_x^\top & 0 & 0 \\ 0 & \mathbf{W}_{ig}^\top & 0 \end{bmatrix} \quad (7.6.1)$$

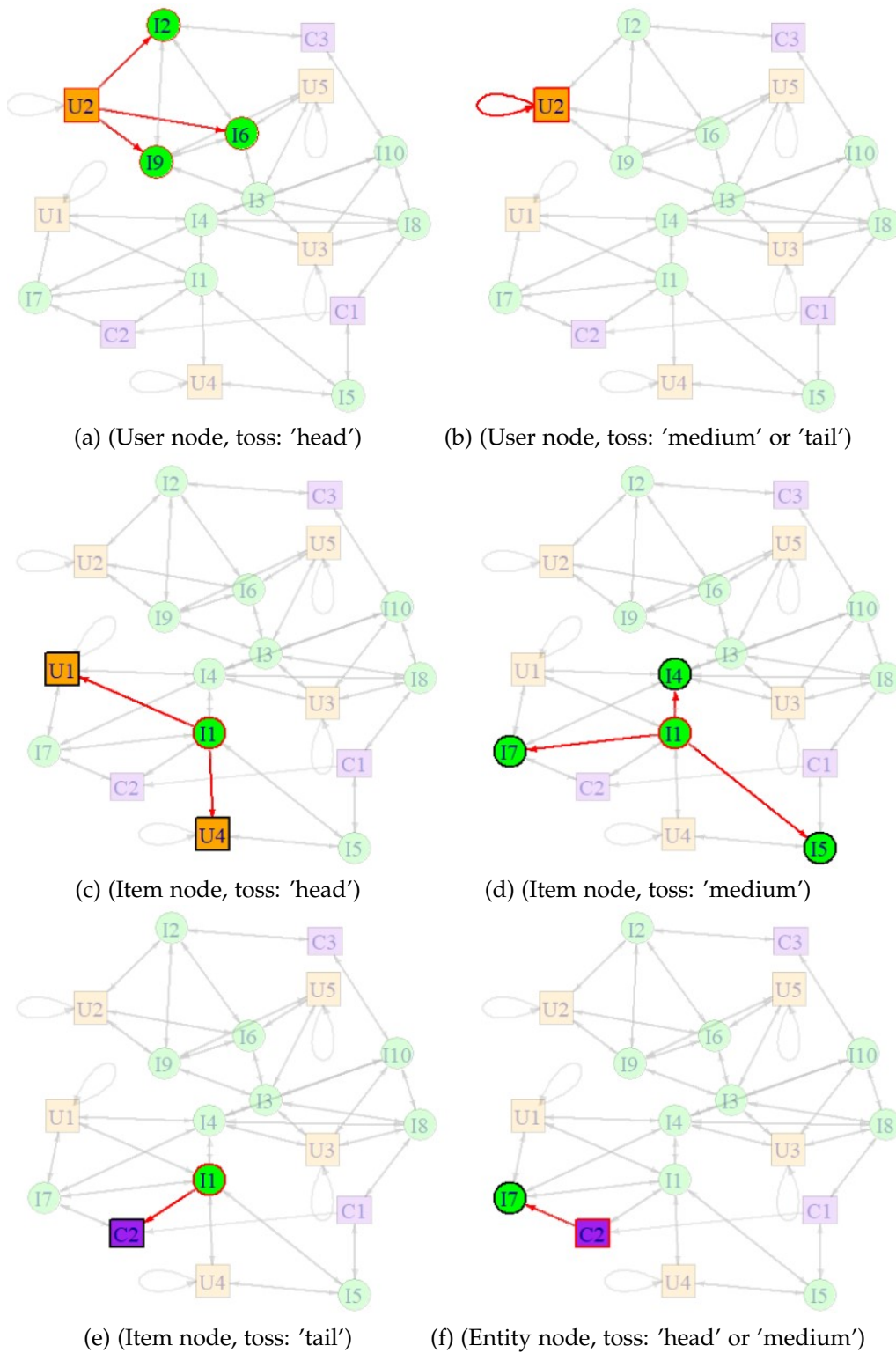


Figure 7.7: KGRECWALK*

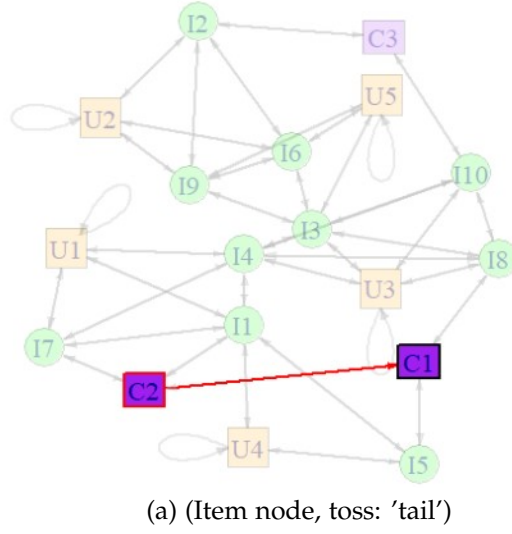


Figure 7.8: KGERECWALK*

$$\mathbf{P} \leftarrow \theta_1 \mathbf{H}_{\text{KGERecWalk}^*} + \theta_2 \begin{bmatrix} \mathbf{I} & 0 & 0 \\ 0 & \mathbf{M}_I & 0 \\ 0 & \mathbf{W}_{ig}^T & 0 \end{bmatrix} + \theta_3 \begin{bmatrix} \mathbf{I} & 0 & 0 \\ 0 & 0 & \mathbf{W}_{ig} \\ 0 & 0 & \mathbf{W}_{kg} \end{bmatrix} \quad (7.6.2)$$

$$\theta_1 + \theta_2 + \theta_3 = 1 \quad (7.6.3)$$

Figures 7.7 and 7.8 both demonstrate the KGERecWalk* working flow for three coin-tossing results. In these two figures, green, orange, and purple nodes represent items, users, and entities.

We use the notations (\mathbf{U} , \mathbf{I} , and \mathbf{G}) to denote all user, item, and entity nodes. Assuming that the walker currently occupies a node ($c \in \mathbf{U} \cup \mathbf{I} \cup \mathbf{G}$), a biased coin-toss determines the next possible move that yields 'Head' with probability (θ_1), 'Medium' with probability (θ_2), and 'Tail' with probability (θ_3). Provided that the current node is a user node ($c \in \mathbf{U}$): i) if the coin-toss yields a head, the walker jumps to one of the item nodes rated by the user randomly (as shown in Figure 7.7 (a), the walker jumps from the user 'U2' to one of the nodes in 'I2', 'I6', and 'I9' randomly.); ii) if the coin-toss yields medium or tail, the walker stays put (as shown in Figure 7.7 (b), the walker stays at the user 'U2'). In another case, the current node (c) is an item node ($c \in \mathbf{I}$): i) if the coin-toss yields a head, the walker randomly jumps to one of the users that have rated the item (as shown in Figure 7.7 (c), the walker randomly jumps from the item node 'I1' to one of the users' nodes 'U1' or 'U4'.); ii) if the coin-toss yields are medium, the walker moves to one of the items that connect with the currently occupied item node (as shown in Figure 7.7 (d), the walker randomly moves from the item node 'I1' to one of the surrounding items nodes 'I4', 'I5', or 'I7'.); iii) if the coin-toss yields tail, the walker moves to an entity node randomly (as shown in Figure 7.7 (e), the walker moves from the item node 'I1' to the entity node 'C2'.). While the current node (c) is an entity node ($c \in \mathbf{G}$): i) if the coin-toss yields a head, the walker jumps to one of the connected item nodes that belong to this category (as shown in Figure 7.7 (f), the

walker moves from the entity node 'C2' to the item node 'I7'.); ii) if the coin-toss yields medium or tail, the walker moves to one of the connected entity node (as shown in Figure 7.8 (a), the walker moves from the item node 'C2' to the item node 'C1').

7.7 CONCLUSION

This chapter mainly discusses the semantic representations of the items in the standard recommendation datasets. As the user-item ratings provided by the standard recommendation datasets did not provide additional semantic information about the users or items, we introduced the knowledge graphs. Section 7.2 mainly discussed the concept and representations of the knowledge graphs. We learned that a knowledge graph contains millions or billions of facts. Each fact within the knowledge graph is considered a triplet of entities and relations. A URI can store and relocate the semantic information of an entity or a relation of the knowledge graph on a webpage.

Subsequently, in section 7.3, we move on to the practical application of knowledge graphs. We start by introducing three standard recommendation datasets (in section 7.3.1). A pivotal step in our process is introducing the standard entity linking technique, which involves mapping the items in the standard recommendation dataset onto the entities in the knowledge graph using the processed URI mapping files.

Following this, we delve into extracting the local knowledge subgraph for items in the standard recommendation dataset in section 7.4. We utilize DBpedia as the source knowledge graph and the DBpedia Mappings Wiki to identify properties most semantically related to the item. These relations are then used as candidate relations, and we employ the DFSPARQL method to extract all n-hop neighbours centred with the item in the source knowledge graph, thereby obtaining the local knowledge graph of the item. We concatenate each item's local knowledge graph as the primary knowledge graph.

However, the original knowledge graph may contain noise, so we need to filter it to obtain a well-structured and clear version, as the knowledge graph filtering technique was discussed in section 7.5. We first introduce the RSCL scheme to classify the semantic relevance between a pair of entities roughly. Then, we use the TransE embedding method to convert entities and relations into embedding vectors. We calculate the similarity of the embedding vectors of a pair of items as their semantic similarity score. Based on the results of RSCL and the embedded semantic similarity scores of the items, we filter out those negative relations from the original knowledge graph, thereby obtaining an accurate and concise filtered knowledge graph that reveals the semantic information about the items.

At the end of this chapter, we propose an extended framework, the KGRecWalk* model, that merges the user-item combination graph and the knowledge graph as a whole graph and modifies the coin-tossing strategy to simulate a surfer's random walk. The KGRecWalk* model considers both user-item interaction information and the items' semantic information.

8

CONCLUSION & FUTURE WORK

8.1 CONCLUSION

8.1.1 Discussion

In this thesis, we have motivated the need to investigate a graph-based personalized recommender system and explored the items' semantic information from knowledge graphs to improve recommendation accuracy.

In Chapter 2, we discussed that a *recommender system* was such a thing used for *information searching* to find out the items the user likes and *information filtering* to filter out the items the user dislikes. Three traditionally used recommender systems include *content-based*, *collaborative*, and *hybrid* filtering. We mainly detailed the neighbour-based methods and model-based methods for the collaborative-filtering. In addition, we showed the neural-based methods. Finally, this Chapter explains hybrid filtering as a combination of content-based and collaborative filtering.

Then, Chapter 3 mainly discussed two graph representations of a *standard recommendation dataset*: a *user-item bipartite* graph and an *inter-item* graph. We also introduced two types of *graph traversals*: *deterministic* and *probabilistic*. We gave DFS as an illustration of the deterministic graph traversals and random walks as examples of the probabilistic graph traversals. We used the random walks on the inter-item graph to make recommendations to the user. At the end of this Chapter, we introduced standard PageRank as an example of random walks. The standard PageRank algorithm is a global node ranking algorithm that assumes each node has the same initial landing probability. As an iterative algorithm, such an algorithm records the landing probability of each node for each iteration and terminates at the convergent state. The items with high landing probabilities are the recommendations to the user. The standard PageRank provides a non-personalized recommendation strategy.

Chapter 4 proposed the Personalized PageRank algorithm, which restrained the *random walks* starting with a small subset of item nodes rated by the user and terminating at the items the user did not interact with. We also introduced *diffusion kernels*, including the *Newmann-series-based* and *exponential-based* kernels, as an effective way to measure the relatedness of a pair of graph nodes. We illustrated how the diffusion kernels work on the inter-item graph.

Chapter 5 discussed the *methodology*, referring to the *RecWalk** model, which is an integration graph of the user-item bipartite graph and the inter-item graph. We extended this model with different diffusion kernels as we listed in Chapter 4, and used this model as a kernel-based rec-

ommendation approach.

Chapter 6 set up the evaluation procedures for the kernel-based recommendation approach. We selected some non-item-based methods (PureSVD, EigenRec, MLP, GMF, and P³) and item-based methods (*Cosine* and *SLIM*) as our baseline recommendation algorithms. We conducted experiments on eight well-known datasets using LOOCV *LOOCV* as our evaluation metric and compared the results of the baseline algorithms and kernel-based recommendation approaches.

Finally, Chapter 7 introduced the item's *semantic information* extraction and transformation. Using the standard entity linking technique, we first linked the items in the standard recommendation dataset onto the entity nodes of the *knowledge graph* (*DBpedia*). Then, we proposed *DFSPARQL* as a deterministic graph traversal to extract the *knowledge subgraph* for each item, and we transformed the knowledge subgraph onto entity *embedding vectors*. At the end of this Chapter, we sketched a method, *KGRecWalk**, that comprises the filtered knowledge subgraph onto the standard *RecWalk** model, as we introduced in Chapter 5.

We now conclude the thesis and discuss some final contributions towards future research.

8.1.2 Thesis Conclusion

Research Question 1: Which diffusion kernel performs best on the standard recommendation graph?

Answer: As we mentioned in Chapter 4, we used the *RecWalk** model combined with different diffusion kernels on eight public, well-known standard recommendation datasets. From their experimental results, we concluded that the Laplacian kernel was superior to other diffusion kernels on both *Cosine* and *SLIM* baseline item models.

Research Question 2: Can we extract a knowledge subgraph using the most relevant relations from the source knowledge graph in order to enhance the recommendation data?

Answer: In Chapter 7, we first used Paolo's item URI mapping profile as the entity linking results. Then, we selected nine relations as the initial candidate relations and proposed the *DFSPARQL* method to search all two-hop neighbours centred on the item URI as the knowledge subgraph based on nine candidate semantically related properties.

Research Question 3: Can the embedding representation be constructed from the knowledge graph to accurately and precisely enhance the semantic representation of the items?

Answer: In Chapter 7, we used *TransE*, a graph embedding technique, to translate the entity nodes of the knowledge graph onto their entity embedding vectors. We then filtered out those negative relations to obtain a filtered knowledge graph that accurately reveals the semantic properties of the items.

8.1.3 Contribution

Overall, this thesis has two significant contributions. The first main contribution is RecWalk*, a graph-based recommender engine that combines a user-item bipartite graph and an inter-item graph. We used random walks, a type of probabilistic graph traversals, on this engine, combining them with the different diffusion kernels to make personalized recommendations to the user.

The second main contribution is the extension of the items' semantic information. We mined them from the source knowledge graph (DBpedia) by extracting a knowledge subgraph for each item based on pre-defined candidate relations using DFSPARQL, a deterministic graph traversal. We then used TransE, a knowledge graph embedding technique, to transform the entities of the knowledge graph onto their entity embedding vectors, and we filtered out those negative relations that bring noises into the knowledge subgraphs.

The secondary contribution is data enrichment using the item's semantic information. We first constructed the embedded-item model using the items' embedding vectors.

8.2 DRAWBACKS

While our RecWalk* model, enhanced with diffusion kernels, has shown promising results by outperforming baseline algorithms on standard and augmented recommendation datasets, there are clear areas that require further development.

Firstly, the RecWalk* model only works with one diffusion kernel at a time. In addition, the RecWalk* model comprises a user-item bipartite graph and an inter-item graph. The user-item bipartite graph represents user-item interactions. The inter-item graph represents the relations between items. As mentioned in Chapter 3, we can construct an inter-item graph in many ways. However, the RecWalk* model only uses the inter-item graph constructed in one way.

In addition, the RecWalk* model connects the user-item bipartite and inter-item graphs with a coin-tossing probability. This probability was a fixed value for the random walker landing at any user but not personalized for each user.

For the items' semantic representation, we only explored two-hop neighbour entities to construct the knowledge subgraph, and we used TransE to transform the entities onto embedding vectors. The TransE embedding method was efficient for one-to-one relations but weak for one-to-many, many-to-one, and many-to-many relations. For instance, a movie has multiple stars and starring is a one-to-many relationship.

By acknowledging the limitations in our proposed framework, we pave the way for future work that will address these issues and further enhance the RecWalk* model.

8.3 FUTURE WORK

Some technical aspects of this work can be improved. In our present work, the RecWalk* model only uses one kernel at a time. We aim to explore a way to combine two or more kernels parallelly on the RecWalk* model.

Additionally, the present RecWalk* model only consists of a user-item bipartite and one inter-item graph. We would extend this model by adding different inter-item graphs simultaneously. For instance, the RecWalk* model can include Cosine and SLIM, two inter-item graphs to customize the tossing probability connecting the user-item bipartite and each inter-item graph.

In our current RecWalk* model, the tossing probability connecting the user-item component and the inter-item component is set as a fixed value shared by all users. However, we see potential in personalizing this probability for each user, and we plan to explore this in the future.

In our current experiments, we have been using the ‘control variates’ strategy to identify the optimal parameters for the RecWalk* model. However, this approach requires us to manually provide a list of candidate values and test each one, which may not fully cover the search space. To address this, we aim to transform the problem into an optimization problem by declaring a loss function to minimize the loss between the actual and predicted values.

Finally, for the items’ semantic representation, as the TransE embedding method was ineffective for one-to-many or many-to-many relations, we would choose a different embedding technique like ‘TransH’ or ‘Node2Vec’.

Appendices

A

UNFILTERED KNOWLEDGE GRAPH

In section A, we plot the candidate relations distribution of the unfiltered knowledge graph for each standard recommendation dataset (ML-300k, FB-MS, and LT, respectively). Some high-frequency candidate relations are significant because they give more generic semantic information for most items; some low-frequency candidate relations only provide the semantic information for the limited items. Therefore, we focus on those high-frequency relations to provide semantic information for more items.

A.1 UNFILTERED ML-300K KNOWLEDGE SUBGRAPH

This section mainly discusses the extraction and construction of the unfiltered ML-300K knowledge graph. We select nine high-quality, clean, and well-constructed film-related relations as initial candidate relations to the ML-300K knowledge graph. These nine candidate relations are 'dcterms: subject', 'skos: broader', 'dbr-owl: starring', 'dbr-owl: writer', 'dbr-owl: producer', 'dbr-owl: distributor', 'dbr-owl: director', 'dbr-owl: author', and 'dbr-owl: composer'. The relations 'dcterms: subject' and 'skos: broader' are essential as they give the categorical information of the film. The rest of the relations are the ontological properties of the film, such as writers, stars, and composers. Table A.1 below gives the statistics information of the unfiltered ML-300K knowledge graph (ML-300K-KG). There are 2907 films among the 3035 films in the database, which are successfully mapped onto the entity nodes in the knowledge graph. ML-300K-KG contains 58,127 semantic entities and 262,587 links.

Dataset	#Links	#Entities	#Covered Items
ML-300K-KG	262,587	58,127	2907/3035

Table A.1: Statistics of the Unfiltered knowledge graph (ML-300K-KG)

In addition, we plot the frequency distribution of the relationship to investigate the importance of each candidate relationship, as shown in Figure A.1 above. In this figure, the x-axis represents each candidate relation, and the y-axis represents the number of occurrences of each candidate relation in the unfiltered knowledge graph. We highlight the top five relations (marked as the red bars) that occurred in the knowledge graph. The top five relations are 'dcterms: subject', 'skos: broader', 'dbr-owl: starring', 'dbr-owl: writer', and 'dbr-owl: producer'. Specifically, 'dcterms: subject' is the most important relation in the unfiltered ML-300K knowledge graph.

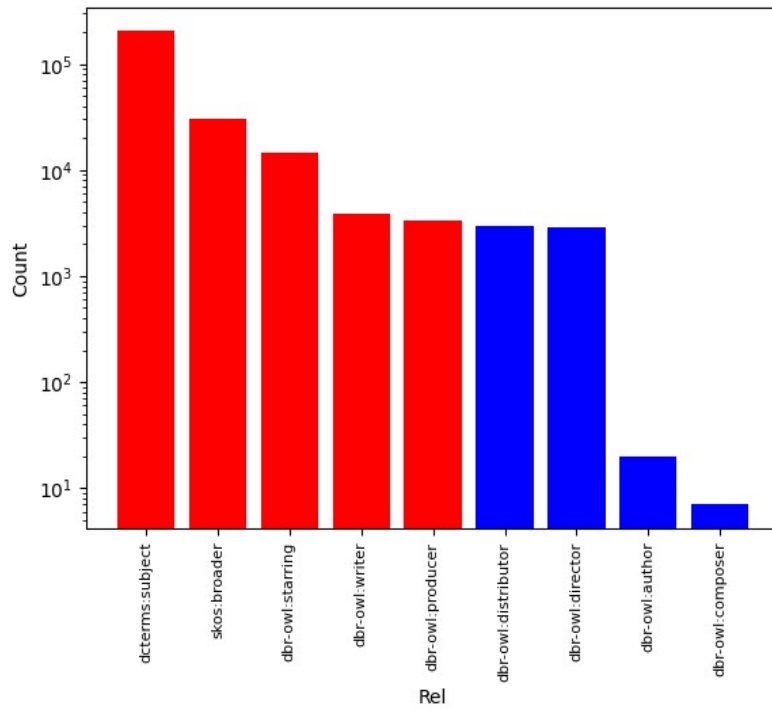


Figure A.1: Relation frequency distribution of the unfiltered ML-300K-KG

A.2 UNFILTERED FB-MS KNOWLEDGE SUBGRAPH

Section A.2 mainly discusses the unfiltered knowledge graph extraction and construction for the FB-MS dataset. We select nine high-quality and well-structured artist-related relations as the initial candidate relations to extract the FB-MS knowledge graph. The nine candidate relations include 'dcterms: subject', 'dbr-owl: genre', 'dbr-owl: associatedMusicalArtist', 'dbr-owl: associatedBand', 'dbr-owl: recordLabel', 'dbr-owl: hometown', 'dbr-owl: formerBandMember', 'dbr-owl: bandmember', and 'dbr-owl: soundRecording'. The 'dcterms: subject' relation gives the categorical information of an artist, and the rest of the candidate relations are the ontological properties related to the artist. Table A.2 below gives the statistics information of the FB-MS knowledge graph. There were 4126 items among 4379 items in the database, which are successfully mapped onto the entity nodes in the knowledge graph. The extracted knowledge contains 61,843 semantic entities and 316,272 links.

Dataset	#Links	#Entities	#Covered Items
FB-MS-KG	316,272	61,843	4126/4379

Table A.2: Statistics of the Unfiltered knowledge graph (FB-MS-KG)

We plot the relation frequency distribution to investigate the importance of the candidate relations, as shown in Figure A.2 below. In this figure, the x-axis represents all candidate relations, and the y-axis represents the occurrence number of each relation in the knowledge graph. We highlight the top five relations (marked as the red bars) in the knowledge graph. The

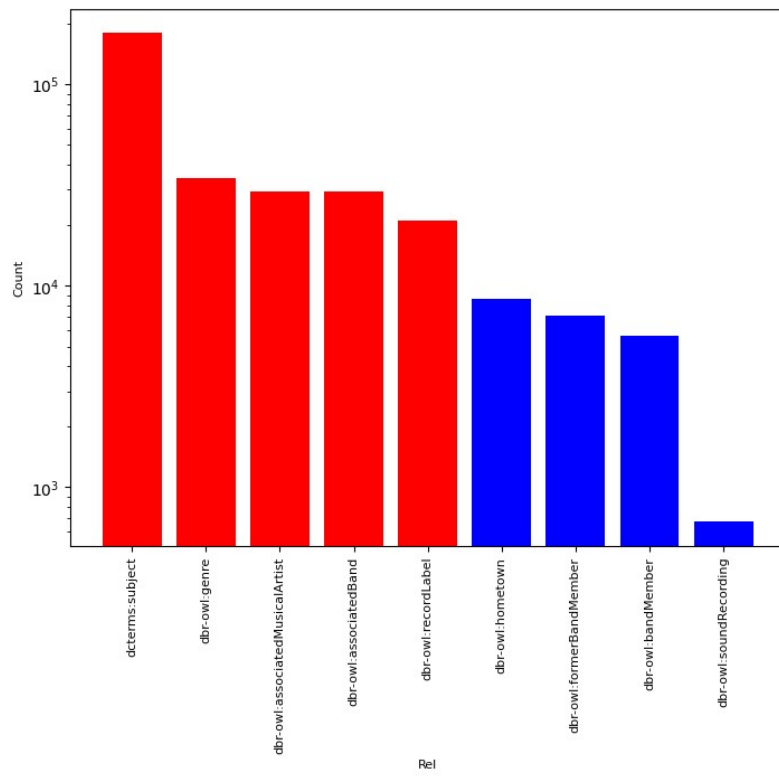


Figure A.2: Relation frequency distribution of the unfiltered FB-MS-KG

top five relations are *'dcterms:subject'*, *'dbr-owl:genre'*, *'dbr-owl:associatedLabelMusicalArtist'*, *'dbr-owl:associatedBand'*, and *'dbr-owl:recordLabel'*.

A.3 UNFILTERED LT KNOWLEDGE SUBGRAPH

Section A.3 mainly discusses the extraction and construction of the unfiltered LT knowledge graph. We select nine high-quality and well-defined book-related properties as the initial candidate relations to extract the LT knowledge graph. The nine candidate relations include 'dc-terms:subject', 'skos:broader', 'dbr-owl:author', 'dbr-owl:writer', 'dbr-owl:illustrator', 'dbr-owl:literaryGenre', 'dbr-owl:mediaType', 'dbr-owl:publisher', and 'dbr-owl:nonFictionSubject'. The relations 'dc-terms:subject' and 'skos:broader' are essential as they give categorical information about the book. The rest of the candidate relations are the ontological properties of the book. Table A.3 below gives the statistics information of the LT knowledge graph (LT-KG). We find that there are 3426 items among 3680 items in the database, which are successfully mapped onto the knowledge graph. The extracted knowledge contains 34,853 semantic entities and 95,619 links.

Dataset	#Links	#Entities	#Covered Items
LT-KG	95,619	34,853	3426/3680

Table A.3: Statistics of the Unfiltered knowledge graph (LT-KG)

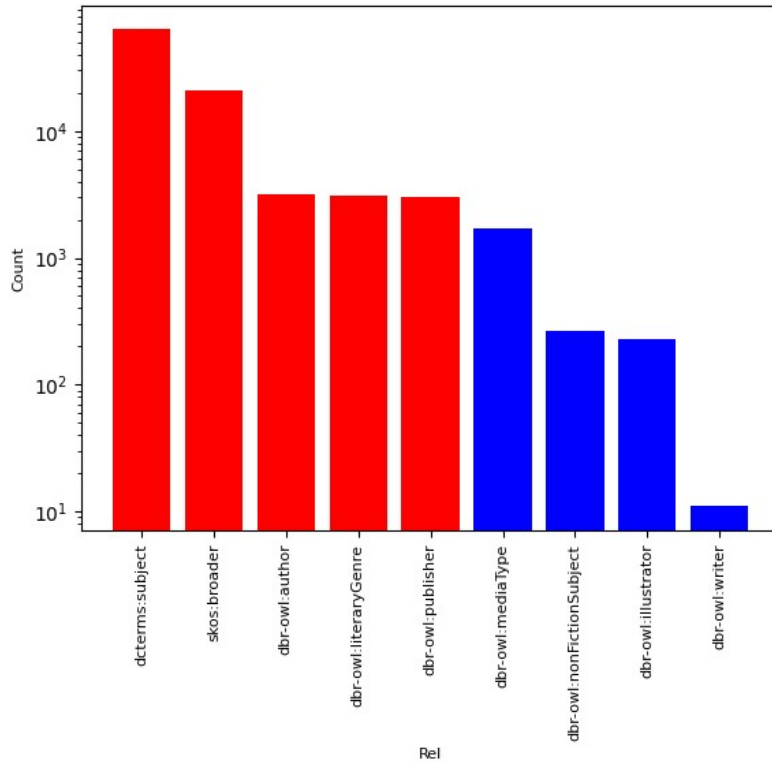


Figure A.3: Relation frequency distribution of the unfiltered LT-KG

We also plot the relation frequency distribution to investigate the importance of relations, as shown in Figure A.3 below. In this figure, the x-axis represents all candidate relations, and the y-axis represents the occurrence number of each relation in the knowledge graph. The relations are ordered by their occurrence number. We highlight the top five relations in the

LT knowledge graph as the approach used in the ML-300K-KG and the FB-MS-KG. The top five relations include 'dcterms:subject', 'skos:broader', 'dbr-owl:author', 'dbr-owl:literaryGenre', and 'dbr-owl:publisher'. We also mark the top five relations as the red bars in the knowledge graph. The top five relations are '*dcterms:subject*', '*dbr-owl:genre*', '*dbr-owl:associatedLabelMusicalArtist*', '*dbr-owl:associatedBand*', and '*dbr-owl:recordLabel*'.

B | FILTERED KNOWLEDGE SUBGRAPH

For each unfiltered knowledge subgraph for the three standard recommendation datasets presented in sections A.1 -A.3, we first choose the top five high-frequency candidate relations to create the filtered knowledge subgraph. Then, we apply the knowledge graph embedding approach to create the embedding vectors for entities. Further, we try different combinations (minimum selecting three relations as one combination) from these five relations to create the filtered knowledge graphs and calculate the semantic similarity scores of the 'Strongly related' and 'Unrelated' item pairs from their embedding vectors. Sections B.1 - B.3 show the optimal relations selection of the filtered knowledge subgraphs for ML-300K, FB-MS, and LT, respectively.

B.1 FILTERED ML-300K KNOWLEDGE SUBGRAPH

This section demonstrates the optimal relations selection of the filtered ML-300K knowledge subgraphs. As shown in Figure A.1 in section A.1, the top-five candidate relations of the primary ML-300K knowledge graph are 'dcterm:subject', 'skos:broader', 'dbr-owl:starring', 'dbr-owl:writer', and 'dbr-owl:producer'. In abbreviation, we map each onto 'subject', 'broader', 'starring', 'writer', and 'producer'.

We first show the statistics information of the candidate-filtered ML-300K knowledge subgraphs by choosing different combinations from the top five candidate relations, as shown in Table B.1 below. In this table, the column 'GID' represents the identifier of each candidate-filtered knowledge graph; the column 'Relations Set' represents the combination of the candidate relations; the columns '#Links' and '#Entities' give the number of links and entities of each filtered knowledge subgraph; and the column '#Covered Items' gives the number of items which are successfully mapped onto the entities in the knowledge graph. Table B.1 includes 15 possible ML-300K candidate-filtered knowledge subgraphs.

We then apply the TransE embedding technique to each candidate-filtered knowledge graph to obtain the embedding vectors for entities and relations. To investigate the best learning rate (lr) and the best embedding size (K) used in the TransE embedding process, we first fix the learning rate ($lr = 0.005$) to test the best embedding size (K). And then, we explore the best learning rate (lr) with the best embedding size. We used the candidate graph ($GID = 1$ in Table B.1) as an example to conduct the experiments and showed the results in Figure B.1. The 'Strongly related' item pairs had the highest average semantic similarity scores ($SimScore$) when the embedding size was 100, and the learning rate was 0.005. Therefore, we used the embedding size ($K = 100$) and learning rate ($lr = 0.005$) as the optimal parameters in the embedding process. We used the same set of parameters to train the embedding vectors for the rest of the candidate-filtered

GID	Relations Set	#Links	#Entities	#Covered Items
1	subject, broader, starring	248,656	54,363	2907
2	subject, broader, writer	238,084	54,238	2907
3	subject, broader, producer	237,565	54,273	2907
4	subject, starring, writer	222,129	44,868	2906
5	subject, starring, producer	221,610	44,909	2906
6	subject, writer, producer	211,038	44,770	2906
7	broader, starring, writer	48,523	28,844	2757
8	broader, starring, producer	48,004	28,209	2751
9	broader, writer, producer	37,432	24,186	2629
10	starring, writer, producer	21,477	9428	2779
11	subject, broader, starring, writer	252,464	54,517	2907
12	subject, broader, starring, producer	251,945	54,558	2907
13	subject, starring, writer, producer	241,373	54,419	2907
14	broader, starring, writer, producer	51,812	30,023	2780
15	subject, broader, starring, writer, producer	255,753	54,698	2907

Table B.1: Statistics of the candidate-filtered ML-300K knowledge subgraphs

knowledge subgraphs.

Table B.2 gives examples of the ‘Strongly related’ movie pairs (POS Examples) and the ‘unrelated’ movie pairs based on their RSCL scores using the number of shared categorical labels (dterms: subject). We ranked all related movie pairs from ‘Strongly related’ to ‘Weakly related’ based on their RSCL scores. For example, the movies Aladdin and The Little Mermaid are the strongest related because they have the highest RSCL score compared with other item pairs. The movies Remember the Titans and The Circus are unrelated because they do not have any shared labels.

Table B.3 shows the average item semantic similarity scores of the related and unrelated movie pairs for each candidate-filtered ML-300K knowledge subgraph. The filtered knowledge subgraph (GID = 2 in Table B.1) has the highest average item semantic similarity scores for all positive movie pairs. Therefore, we selected this graph as the optimal filtered knowledge subgraph, and the optimal relations are ‘*subject*’, ‘*broader*’, and ‘*writer*’.

POS Examples	Movie-Pair Information	RSCL
Strongly Related	Aladdin (1992 film) The Little Mermaid (1989 film)	100%
Strongly Related	Free Willy Free Willy 2: The Adventure Home	96.8%
Strongly Related	Teenage Mutant Ninja Turtles (1990 film) & Teenage Mutant Ninja Turtles II: The Secret of the Ooze	83.9%
...
NEG Examples	Movie-Pair Information	RSCL
Unrelated	Remember the Titans & The Circus (film)	0
Unrelated	Stepmom (film) & Grandview, U.S.A	0
Unrelated	Breaking Away & Ugly Aur Pagli	0
...

Table B.2: Semantic related and unrelated movie-pairs

GID	POS Examples	NEG Examples
	Mean	Mean
1	0.664	0.341
2	0.712	0.308
3	0.598	0.306
4	0.641	0.333
5	0.666	0.327
6	0.695	0.296
7	0.096	0.062
8	0.089	0.06
9	0.081	0.044
10	0.094	0.058
11	0.643	0.346
12	0.653	0.345
13	0.623	0.317
14	0.085	0.062
15	0.697	0.352

Table B.3: Average item semantic similarity scores of the related and unrelated item pairs of each candidate filtered ML-300K knowledge subgraph

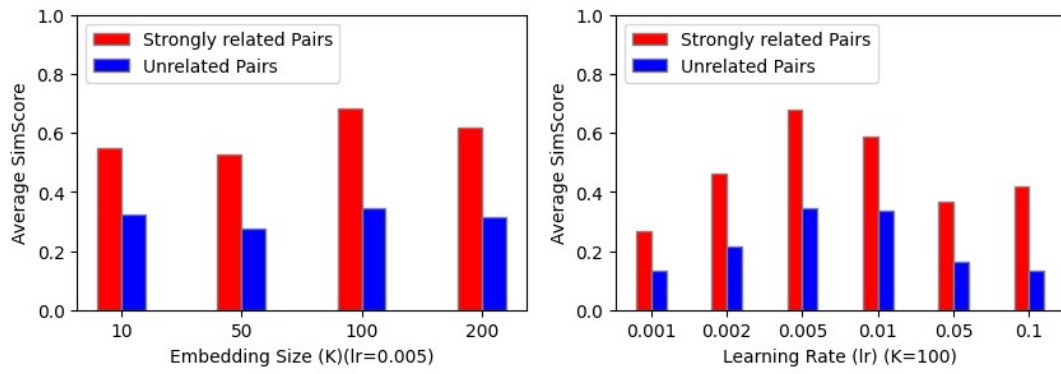


Figure B.1: Parameters Selection: embedding size (K) and the learning rate (r) (Filtered ML-300K Sub-KG)

At the end of this section, we show a snapshot of the local knowledge subgraph for the movie 'Touch of Evil', as shown in Figure B.2. In this figure, the green node in the centre represents the movie; the blue node represent the writers or stars like 'Orson Welles'; and the red nodes are the categorical information about the people or movie.



Figure B.2: Local knowledge subgraph demonstration for the movie ‘Touch of Evil’

B.2 FILTERED FB-MS KNOWLEDGE SUBGRAPH

This section demonstrates the optimal relations selection of the filtered FB-MS knowledge subgraph. As shown in Figure A.2 in section A.2, the top-five candidate relations of the unfiltered FB-MS knowledge subgraph are ‘*dcterms:subject*’, ‘*dbr-owl:genre*’, ‘*dbr-owl:associatedMusicalArtist*’, ‘*dbr-owl:associatedBand*’, and ‘*dbr-owl:recordLabel*’. In abbreviation, we map each onto ‘*subject*’, ‘*genre*’, ‘*associatedMusicalArtist*’, ‘*associatedBand*’, and ‘*recordLabel*’, respectively.

We first show the statistics information of the candidate-filtered ML-300K knowledge subgraphs by choosing different combinations (at least three relations as one combination) from the top five candidate relations, as shown in Table B.4 below. Table B.4 includes 15 candidate-filtered knowledge subgraphs.

We then apply the TransE embedding technique to each candidate-filtered knowledge graph to obtain the embedding vectors for entities and relations. First, we fix the learning rate ($lr = 0.005$) to test the best embedding size (K). Then, we explore the best learning rate (lr) with the best embedding size. We used the candidate-filtered knowledge graph ($GID = 1$ in Table B.4) as an example to conduct the experiments and showed the results in Figure B.3.

We found that the strongly related artist pairs had the highest average semantic similarity scores when the embedding size (K) was 50, and the learning rate (lr) was 0.005. Thus, we used the embedding size ($K = 50$) and the learning rate ($lr = 0.005$) as the optimal parameters in the TransE embedding process to learn the embedding vectors of entities and relations for the rest of the filtered knowledge subgraphs.

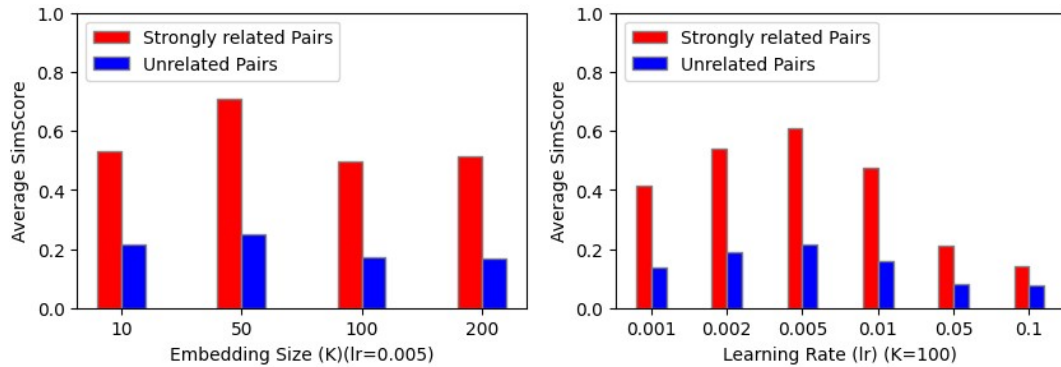


Figure B.3: Parameters Selection: embedding size (K) and the learning rate (r) (Filtered FB-MS Sub-KG)

Table B.5 gives examples of the ‘related’ artist pairs (POS Examples) and the ‘unrelated’ artist pairs (NEG Examples) based on their RSCL scores using the number of shared categorical labels (*dcterms: subject*). We first ranked all related artist pairs from ‘Strongly related’ to ‘Weakly related’ based on their RSCL scores. For example, the artists *Jennifer Lopez* and *Christina Aguilera* emerged as the strongest semantic-related artists because they had the highest RSCL score compared with other item pairs. The artists ‘Mike Posner’ and ‘Tiger Army’ are unrelated as they have no common subjects.

GID	Relations Set	#Links	#Entities	#Covered Items
1	subject, genre, associatedMusicalArtist	243,639	55,015	4126
2	subject, genre, associatedBand	243,636	55,011	4126
3	subject, genre, recordLabel	235,094	49,360	4080
4	subject, associatedMusicalArtist, associatedBand	238,887	54,229	4126
5	subject, associatedBand, dbr-owl:recordLabel	230,345	56,126	4126
6	subject, associatedMusicalArtist, recordLabel	230,342	56,122	4126
7	genre, associatedMusicalArtist, associatedBand	93,504	17,648	3777
8	genre, associatedMusicalArtist, recordLabel	84,962	20,438	3797
9	genre, associatedBand, recordLabel	84,959	20,435	3797
10	associatedMusicalArtist, associatedBand, recordLabel	80,210	18,688	3690
11	subject, genre, associatedMusicalArtist, associatedBand	273,222	55,016	4126
12	subject, genre, associatedMusicalArtist, recordLabel	264,680	56,913	4126
13	subject, associatedMusicalArtist, associatedBand, recordLabel	259,928	56,127	4126
14	genre, associatedMusicalArtist, associatedBand, recordLabel	114,545	20,440	3797
15	subject, genre, associatedMusicalArtist, associatedBand, recordLabel	294,263	56,914	4126

Table B.4: Statistics of the candidate-filtered FB-MS knowledge subgraphs

Table B.6 shows the average items' semantic similarity scores of the related and unrelated item pairs of each candidate-filtered FB-MS knowledge subgraph. The filtered FB-MS knowledge subgraph (GID=11 in Table B.4) had the highest average embedded-item similarity scores for all related item pairs. Therefore, we select this graph as the optimal filtered knowledge subgraph, and the optimal relations are '*subject*', '*genre*', '*associatedBand*', and '*associatedMusicalArtist*'.

POS Examples	Artist-Pair Information	RSCL
Strongly Related	Jennifer Lopez & Christina Aguilera	100%
Strongly Related	Jay-Z & Sean Combs	87.5%
Strongly Related	Janet Jackson & Christina Aguilera	85%
...
NEG Examples	Artist-Pair Information	RSCL
Unrelated	Mike Posner & Tiger Army	0
Unrelated	Morcheeba & Hellyeah	0
Unrelated	The Mars Volta & Apparatus	0
...

Table B.5: Semantic related and unrelated artist-pairs

At the end of this section, we show a snapshot of the locally extracted knowledge subgraph for the artist *Sarah Blasko*, as shown in Figure B.4. In this figure, the blue node in the centre represents the artist; the purple nodes represent the genres, like 'Pop rock' or 'Folk music'; the brown nodes represent the associated bands or artists like 'Dirty Three' or 'Seeker Lover Keeper'; and the red nodes are the categorical information about the artist.

GID	POS Examples	NEG Examples
	Mean	Mean
1	0.709	0.25
2	0.685	0.245
3	0.629	0.262
4	0.705	0.22
5	0.656	0.252
6	0.677	0.254
7	0.101	0.077
8	0.11	0.102
9	0.062	0.102
10	0.074	0.076
11	0.63	0.234
12	0.688	0.266
13	0.654	0.232
14	0.05	0.1
15	0.492	0.246

Table B.6: Average item semantic similarity scores of the related and unrelated item pairs of each candidate filtered FB-MS knowledge subgraph



Figure B.4: Local knowledge subgraph demonstration for the artist 'Sarah Blasko'

B.3 FILTERED LT KNOWLEDGE SUBGRAPH

This section illustrates the selection of optimal relations of the filtered LT knowledge subgraph. As shown in Figure A.3 in section A.3 the top-five candidate relations of the unfiltered LT knowledge subgraph are *'dcterms:subject'*, *'skos:broader'*, *'dbr-owl:author'*, *'dbr-owl:literaryGenre'*, and *'dbr-owl:publisher'*. In abbreviation, we map them onto *'subject'*, *'broader'*, *'author'*, *'literaryGenre'*, and *'publisher'*, respectively.

Like the filtered ML-300K knowledge and the filtered FB-MS knowledge graph, we first show the statistics information (as shown in Table B.7) of each candidate-filtered LT knowledge graph by choosing different combinations (at least three relations as a combination) from the top five candidate relations in the unfiltered LT knowledge subgraph.

Table B.8 gives examples of the 'related' book pairs (POS Examples) and the 'unrelated' book pairs (NEG Examples) based on their RSCL scores using the number of shared categorical labels (*dcterms: subject*). We first ranked all related book pairs from 'Strongly related' to 'Weakly related' based on their RSCL scores.

For example, the books *'Bridget Jones: The Edge of Reason'* and *'Bridget Jones's Diary'* are the strongest semantic-related books because they have the highest RSCL score compared with other book pairs. The book pairs are unrelated if they do not have any shared labels.

We then apply the TransE embedding technique to each candidate-filtered knowledge graph to obtain the embedding vectors for entities and relations. Like the ML-300K and the FB-MS knowledge graph, we first fix the learning rate ($lr = 0.005$) to test the best embedding size (K). Then, we explore the best learning rate (lr) with the best embedding size. We used the candidate graph ($GID = 1$ in Table B.7) as an example to conduct the experiments and showed the results in Figure B.5. We found that the related book pairs have the highest average semantic similarity scores when the embedding size (K) was 50, and the learning rate (lr) was 0.005. Thus, we used the embedding size ($K = 50$) and the learning rate ($lr = 0.005$) as the optimal parameters in the TransE embedding process to learn the embedding vectors of entities and relations for the rest of the candidate-filtered knowledge graphs.

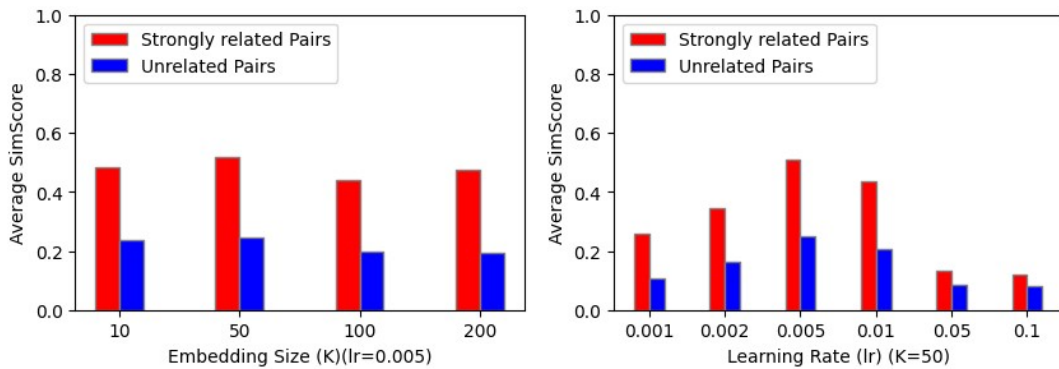


Figure B.5: Parameters Selection: embedding size (K) and the learning rate (r) (Filtered LT Sub-KG)

GID	Relations Set	#Links	#Entities	#Covered Items
1	subject, broader, author	89,407	31,117	3385
2	subject, broader, literaryGenre	89,309	31,199	3384
3	subject, broader, publisher	89,193	31,323	3378
4	subject, author, literaryGenre	71,767	23,244	3385
5	subject, author, publisher	71,651	23,368	3386
6	subject, literaryGenre, publisher	71,553	23,450	3386
7	broader, author, literaryGenre	27,536	16,939	3288
8	broader, author, publisher	27,420	17,163	3300
9	broader, literaryGenre, publisher	27,322	16,161	3300
10	author, literaryGenre, publisher	9780	2415	3304
11	subject, broader, author, literaryGenre	92,673	31,280	3385
12	subject, broader, author, publisher	92,557	31,404	3386
13	subject, broader, literaryGenre, publisher	92,459	31,486	3386
14	subject, author, literaryGenre, publisher	74,917	23,531	3386
15	subject, broader, author, literaryGenre, publisher	95,823	31,567	3386

Table B.7: Statistics of the candidate-filtered LT knowledge subgraphs

POS Examples	Book-Pair Information	RSCL
Strongly Related	Bridget Jones: The Edge of Reason & Bridget Jone’s Diary	100%
Strongly Related	Nineteen Eighty-Four & Animal Farm	68.4%
Strongly Related	Jurassic Park & The Lost World	68.4%
...
NEG Examples	Book-Pair Information	RSCL
Unrelated	The Crystal Cave & The Long Hard Road Out of Hell	0
Unrelated	Jonathan Strange & Mr Norrell & Human, All Too Human	0
Unrelated	The Twits & My Name is Red	0
...

Table B.8: Semantic related and unrelated book-pairs

Table B.9 shows the average items’ semantic similarity scores of positive related and unrelated book-pairs of the filtered LT knowledge graph. The filtered LT knowledge graph ($GID=12$ in Table B.7) has the highest average similarity score for positively related item pairs. Therefore, we selected this graph as the optimal filtered knowledge subgraph, and the optimal set of relations are ‘*subject*’, ‘*broader*’, ‘*author*’, and ‘*publisher*’.

At the end of this section, we show a snapshot of the locally extracted knowledge subgraph of the book ‘The Horse and His Boy’, as shown in Figure B.6. In this Figure, the green node in the centre represents the book ‘The Horse and His Boy’; the brown node represents the author, such as C.S. Lewis, or the publisher, such as Geoffrey Bles; and the red nodes are the categorical information about the book.

GID	POS Examples	NEG Examples
	Mean	Mean
1	0.518	0.247
2	0.537	0.230
3	0.449	0.236
4	0.497	0.231
5	0.523	0.230
6	0.581	0.218
7	0.093	0.069
8	0.105	0.068
9	0.1	0.069
10	0.083	0.065
11	0.478	0.250
12	0.605	0.251
13	0.527	0.238
14	0.559	0.231
15	0.514	0.252

Table B.9: Average item semantic similarity scores of the related and unrelated item pairs of each candidate filtered LT knowledge subgraph

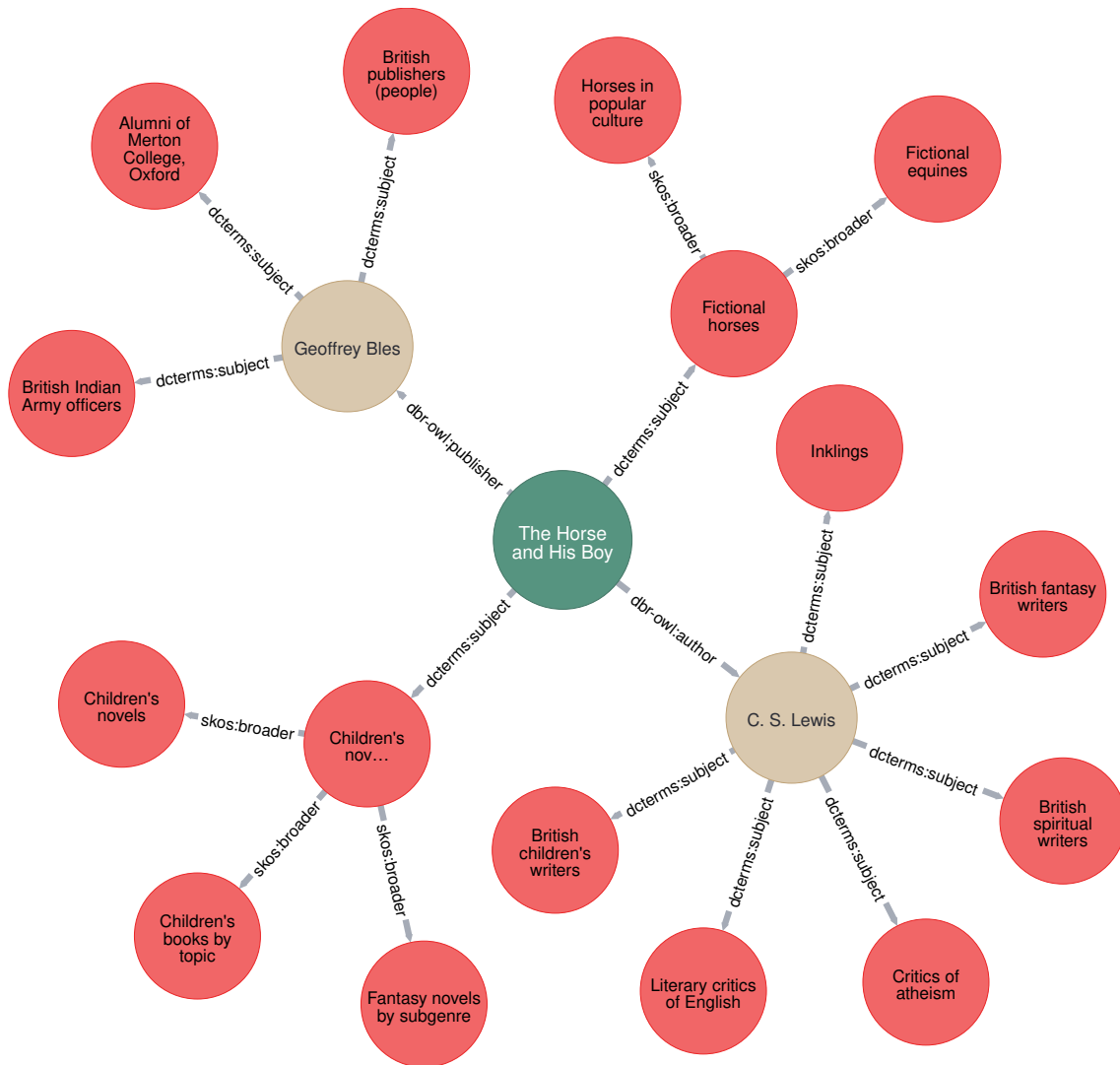


Figure B.6: Local knowledge graph demonstration for the book 'The Horse and His Boy'

BIBLIOGRAPHY

- [ABK+07] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives. “Dbpedia: A Nucleus for a Web of Open Data”. In: *International Semantic Web Conference*. Springer. 2007, pp. 722–735.
- [ACR17] K. Avrachenkov, P. Chebotarev, and D. Rubanov. “Kernels on Graphs as Proximity Measures”. In: *International Workshop on Algorithms & Models for the Web-graph*. 2017.
- [BA11] T. Bogers and V. D. B. Antal. “Introduction to Recommender Systems Handbook”. In: (2011).
- [Bap10] R. B. Bapat. *Graphs and matrices*. Vol. 27. Springer, 2010.
- [BFM04] T. Berners-Lee, R. Fielding, and L. Masinter. “Uniform resource identifier”. In: *Copyright (C) The Internet Society,[Online]. Retrieved from the Internet;,(Jan. 2005) 57* (2004).
- [BMV+12] B. Bahmani, B. Moseley, A. Vattani, R. Kumar, and S. Vassilvitskii. “Scalable k-means++”. In: *arXiv preprint arXiv:1203.6402* (2012).
- [BUG+13] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko. “Translating embeddings for modeling multi-relational data”. In: *Advances in neural information processing systems* 26 (2013).
- [BW84] A. Bundy and L. Wallen. “Breadth-First Search”. In: *Catalogue of artificial intelligence tools* (1984), pp. 13–13.
- [Caw06] G. C. Cawley. “Leave-one-out cross-validation based model selection criteria for weighted LS-SVMs”. In: *The 2006 IEEE international joint conference on neural network proceedings*. IEEE. 2006, pp. 1661–1668.
- [Cha21] B. Chan. *Semantic Search with Milvus, Knowledge Graph QA, Web Crawlers and more!* <https://medium.com/deepset-ai/semantic-search-with-milvus-knowledge-graph-qa-web-crawlers-and-more>. 2021.
- [Chuo7] F. Chung. “The Heat Kernel as the Pagerank of a Graph”. In: *Proceedings of the National Academy of Sciences* 104.50 (2007), pp. 19735–19740.
- [CKT10] P. Cremonesi, Y. Koren, and R. Turrin. “Performance of Recommender Algorithms on Top-N Recommendation Tasks”. In: *Proceedings of the fourth ACM conference on Recommender systems*. 2010, pp. 39–46.
- [CLS90] D. S. Cohen, T. Lee, and D. Sklar. “Precalculus, A Problems-Oriented Approach”. In: (1990).
- [DMO+12] T. Di Noia, R. Mirizzi, V. C. Ostuni, D. Romito, and M. Zanker. “Linked Open Data to Support Content-based Recommender Systems”. In: *Proceedings of the 8th International Conference on Semantic Systems*. I-SEMANTICS '12. Graz, Austria: ACM, 2012, pp. 1–8. ISBN: 978-1-4503-1112-0. DOI: [10.1145/2362499.2362501](https://doi.org/10.1145/2362499.2362501). URL: <http://doi.acm.org/10.1145/2362499.2362501>.

- [ERK+11] M. D. Ekstrand, J. T. Riedl, J. A. Konstan, et al. "Collaborative Filtering Recommender Systems". In: *Foundations and Trends® in Human-Computer Interaction* 4.2 (2011), pp. 81–173.
- [Eve11] S. Even. *Graph Algorithms*. Cambridge University Press, 2011.
- [Fan21] A. L. D. Fante. *Understanding Recommender Systems*. 2021. URL: <https://arturlunardi.medium.com/understanding-recommender-systems-1077f4215516>.
- [Felo5] C. Fellbaum. "Wordnet and Wordnets". In: *Encyclopedia of Language and Linguistics*. Ed. by K. Brown. Elsevier, 2005, pp. 2–665.
- [FŞA+20] D. Fensel, U. Şimşek, K. Angele, E. Huaman, E. Kärle, O. Panasiuk, I. Toma, J. Umbrich, A. Wahler, D. Fensel, et al. "Introduction: What Is a Knowledge Graph?" In: *Knowledge graphs: Methodology, tools and selected use cases* (2020), pp. 1–10.
- [GFS+09] J. Gao, W. Fan, Y. Sun, and J. Han. "Heterogeneous Source Consensus Learning via Decision Propagation and Negotiation". In: *Acm Sigkdd International Conference on Knowledge Discovery & Data Mining*. 2009.
- [Goo24] I. Google. *Machine Learning | Recommendation | Content-based*. <https://developers.google.com/machine-learning/recommendation/content-based/basics?hl=zh-cn> [Accessed: (Use the date of access)]. 2024.
- [Gou13] R. Gould. *Graph Theory*. Courier Corporation, 2013.
- [GPR+07] M. Gori, A. Pucci, V. Roma, and I. Siena. "Itemrank: A random-walk based scoring algorithm for recommender engines." In: *IJCAI*. Vol. 7. 2007, pp. 2766–2771.
- [Gro65] B. M. Gross. "The Managing of Organizations : The Administrative Struggle". In: *American Sociological Review* 30 (1965), p. 606.
- [HA]+22] L. Ha, M. H. Abuljadail, C. Y. Joa, and K. Kim. "Personalized vs non-personalized recommendations: how recommender systems, recommendation sources and recommendation platforms affect trial of YouTube videos among digital natives in Saudi Arabia". In: *Journal of Islamic Marketing* 13.12 (2022), pp. 2778–2797.
- [HK15] F. M. Harper and J. A. Konstan. "The MovieLens Datasets: History and Context". In: *ACM Trans. Interact. Intell. Syst.* 5.4 (Dec. 2015). ISSN: 2160-6455. DOI: [10.1145/2827872](https://doi.org/10.1145/2827872). URL: <https://doi.org/10.1145/2827872>.
- [HLZ+17] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua. "Neural Collaborative Filtering". In: *Proceedings of the 26th international conference on world wide web*. 2017, pp. 173–182.
- [HM16] R. He and J. McAuley. "Ups and Downs: Modeling the Visual Evolution of Fashion Trends with One-Class Collaborative Filtering". In: *International World Wide Web Conferences Steering Committee* (2016).
- [Hol21] J. Holze. *DBpedia Snapshot 2021-09 Release - DBpedia Association*. <https://www.dbpedia.org/blog/snapshot-2021-09-release/>. 2021.
- [HRN+13] B. Hachey, W. Radford, J. Nothman, M. Honnibal, and J. R. Curran. "Evaluating Entity Linking with Wikipedia". In: *Artificial intelligence* 194 (2013), pp. 130–150.
- [JPLo4] K.-Y. Jung, D.-H. Park, and J.-H. Lee. "Hybrid Collaborative Filtering and Content-Based Filtering for Improved Recommender System". In: *International Conference on Computational Science*. Springer. 2004, pp. 295–302.

- [JSK10] G. Jawaheer, M. Szomszor, and P. Kostkova. "Comparison of Implicit and Explicit Feedback From an Online Music Recommendation Service". In: *proceedings of the 1st international workshop on information heterogeneity and fusion in recommender systems*. 2010, pp. 47–51.
- [KAU16] S. Khusro, Z. Ali, and I. Ullah. "Recommender Systems: Issues, Challenges, and Research Opportunities". In: *Information science and applications (ICISA) 2016*. Springer. 2016, pp. 1179–1189.
- [KBV09] Y. Koren, R. Bell, and C. Volinsky. "Matrix Factorization Techniques for Recommender Systems". In: *Computer* 42.8 (2009), pp. 30–37.
- [KL02] R. I. Kondor and J. Lafferty. "Diffusion Kernels on Graphs and Other Discrete Structures". In: *Proceedings of the 19th international conference on machine learning*. Vol. 2002. 2002, pp. 315–322.
- [KLP+06] B. M. Kim, Q. Li, C. S. Park, S. G. Kim, and J. Y. Kim. "A new approach for combining content-based and collaborative filters". In: *Journal of Intelligent Information Systems* 27 (2006), pp. 79–91.
- [KSS+20] A. Kumar, S. S. Singh, K. Singh, and B. Biswas. "Link Prediction Techniques, Applications, and Performance: A Survey". In: *Physica A: Statistical Mechanics and its Applications* 553 (2020), p. 124289.
- [LIJ+15] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morse, P. Van Kleef, S. Auer, et al. "Dbpedia—A large-scale, Multilingual Knowledge Base Extracted from Wikipedia". In: *Semantic web* 6.2 (2015), pp. 167–195.
- [Lik32] R. Likert. "A technique for the measurement of attitudes." In: *Archives of psychology* (1932).
- [LM06] A. N. Langville and C. D. Meyer. *Google's PageRank and Beyond: The Science of Search Engine Rankings*. Princeton university press, 2006.
- [NK11] X. Ning and G. Karypis. "SLIM: Sparse Linear Methods for Top-N Recommender Systems". In: *2011 IEEE 11th international conference on data mining*. IEEE. 2011, pp. 497–506.
- [NK19] A. N. Nikolakopoulos and G. Karypis. "RecWalk: Nearly Uncoupled Random Walks for Top-N Recommendation". In: *Proceedings of the twelfth ACM international conference on web search and data mining*. 2019, pp. 150–158.
- [NOT+16] T. D. Noia, V. C. Ostuni, P. Tomeo, and E. D. Sciascio. "Sprank: Semantic path-based ranking for top-n recommendations using linked open data". In: *ACM Transactions on Intelligent Systems and Technology (TIST)* 8.1 (2016), pp. 1–34.
- [PB07] M. J. Pazzani and D. Billsus. "Content-Based Recommendation Systems". In: *The Adaptive Web: Methods and Strategies of Web Personalization*. Springer, 2007, pp. 325–341.
- [Pea05] K. Pearson. "The Problem of the Random Walk". In: *Nature* 72.1867 (1905), pp. 342–342.
- [Rom22] M. B. Roman. *Steam Games Dataset*. 2022. DOI: [10.34740/KAGGLE/DS/2109585](https://doi.org/10.34740/KAGGLE/DS/2109585). URL: <https://www.kaggle.com/ds/2109585>.

- [RRS10] F. Ricci, L. Rokach, and B. Shapira. "Introduction to Recommender Systems Handbook". In: *Recommender systems handbook*. Springer, 2010, pp. 1–35.
- [RSH+16] T. Rebele, F. Suchanek, J. Hoffart, J. Biega, E. Kuzey, and G. Weikum. "YAGO: a Multilingual Knowledge Base from Wikipedia, Wordnet, and Geonames". In: *Springer International Publishing* (2016).
- [SCo4] J. Shawe-Taylor and N. Cristianini. "Kernel Methods for Pattern Analysis". In: *publications of the american statistical association* (2004).
- [Sen96] E. Seneta. "Markov and the Birth of Chain Dependence Theory". In: *International Statistical Review/Revue Internationale de Statistique* (1996), pp. 255–263.
- [SKK+01] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. "Item-Based Collaborative Filtering Recommendation Algorithms". In: *Proceedings of the 10th international conference on World Wide Web*. 2001, pp. 285–295.
- [Spi12] D. Spielman. "Spectral Graph Theory". In: *Combinatorial scientific computing* 18 (2012), p. 18.
- [SS15] L. Schjoedt and K. Sangboon. "Control variables: problematic issues and best practices". In: *The Palgrave handbook of research design in business and management*. Springer, 2015, pp. 239–261.
- [Tho53] R. L. Thorndike. "Who belongs in the family?" In: *Psychometrika* 18.4 (1953), pp. 267–276.
- [Tor20] P. Torres-Tramón. "Diffusion-based models for semantic relatedness". PhD thesis. NUI Galway, 2020.
- [Tri24] I. Triply. *The network effect for your data - Triply*. <https://triplly.cc/>. 2024.
- [WCL+23] R. Widayanti, M. H. R. Chakim, C. Lukita, U. Rahardja, and N. Lutfiani. "Improving Recommender Systems using Hybrid Techniques of Collaborative Filtering and Content-Based Filtering". In: *Journal of Applied Data Sciences* 4.3 (2023), pp. 289–302.
- [WMC+18] C. Wang, X. Ma, J. Chen, and J. Chen. "Information Extraction and Knowledge Graph Construction From Geoscience Literature". In: *Computers & geosciences* 112 (2018), pp. 112–120.
- [WMW+17] Q. Wang, Z. Mao, B. Wang, and L. Guo. "Knowledge Graph Embedding: A Survey of Approaches and Applications". In: *IEEE transactions on knowledge and data engineering* 29.12 (2017), pp. 2724–2743.
- [ZH22] Y. Zhou and C. Hayes. "Graph-Based Diffusion Method for Top-N Recommendation". In: *Irish Conference on Artificial Intelligence and Cognitive Science*. Springer, 2022, pp. 292–304.
- [ZMK+05] C. N. Ziegler, S. M. Mcnee, J. A. Konstan, and G. Lausen. "Improving Recommendation Lists Through Topic Diversification". In: *International Conference on World Wide Web*. 2005, p. 22.