University of Zurich[UZH]
Department of Informatics

Recommending Long-Tail Items with Short Random Walks over the User-Item-Feedback Graph

Master's Thesis
December 26, 2014

Fabian L. Christoffel
of Zurich ZH, Switzerland

Student-ID: 05-729-686
fabian.christoffel@uzh.ch

Advisor: Chris Newell, DPhil
British Broadcasting Corporation
Research & Development
http://www.bbc.co.uk/rd

Co-Advisor: Bibek Paudel
Department of Informatics
University of Zurich
http://www.ifi.uzh.ch/ddis

Prof. Abraham Bernstein, PhD
Department of Informatics
University of Zurich
http://www.ifi.uzh.ch/ddis

Title page image: Drawing of a Sargassum fish from A. H. Baldwin in "The Bahama Islands" by The Geographical Society of Baltimore, 1905. Available from *commons.wikimedia.org/wiki/File:Histrio_histrio_by_A._H._Baldwin.jpg*

Acknowledgements

I would like to express my appreciation and thanks to my supervisor Professor Dr. Abraham Bernstein for his guidance and the very fruitful discussions, but also for his efforts to arrange a collaboration for this project with the British Broadcasting Corporation (BBC). Equally important for the success of the project was my advisor Dr. Chris Newell of the Research and Development (R&D) department of the BBC. Thank you Chris for your very helpful advice and strong support during the entire project. I am very grateful that you agreed to this collaboration and hope that the results of the research are beneficial for your future work. A great impact on the outcome of the project had also Bibek Paudel, who helped me to understand the research field and greatly improved the thesis with his review comments. Thank you for your patience, Bibek. Furthermore I would like to thank all the staff members of BBC R&D for the interesting discussions and their welcoming attitude. I am also very grateful for the enduring and manifold support throughout my studies of my parents, sister, and friends. This work was made possible with financial support provided by the BBC.

Zusammenfassung

Wir untersuchen Rangfolgealgorithmen von Graphknoten für die Anwendung in Empfehlungssystemen basierend auf kollaborativem Filtern. In dieser Arbeit evaluieren wir die Leistung früher vorgeschlagenen Rangfolgealgorithmen mit Experimenten an vier historischen Datensätzen bestehend aus impliziten Artikelbewertungen. Zusätzlich zu der Vorhersagekraft bezüglich des zukünftigen Benutzerverhaltens (Richtigkeit) beurteilen wir die Qualität der Empfehlungslisten anhand von vier weiteren Leistungskriterien: Unterschiedlichkeit der vorgeschlagenen Artikel, Katalogabdeckung, Personalisierung und Neuigkeit/Überraschungswert. Die Ergebnisse unserer Experimente zeigen, dass Empfehlungslisten die mit Rangfolgealgorithmen von Graphknoten erstellt werden, von den populärsten Artikeln dominiert werden. Die Empfehlungslisten weisen schlechtere Werte auf für Richtigkeit, Katalogabdeckung, Personalisierung und Neuigkeit/Überraschung als Listen von Empfehlungssystemen, die auf Nächste-Nachbarn-Klassifizierung oder einem latent factor model basieren.

Mit einem parametrisierten Umrangierungsverfahren das populäre Artikel tiefer gewichtet als unpopuläre Artikel, angewendet auf Rangfolgealgorithmen basierend auf der Knotenübergangswahrscheinlichkeit in einem random walk (insbesondere P^3 und P^5 [Cooper et al., 2014]), erzielten wir höhere Neuigkeit/Überraschungswerte und eine bessere Katalogabdeckung und Personalisierung. Für eine mässig starke Untergewichtung von beliebten Artikeln im Umrangierungsverfahren erhöhte sich die Richtigkeit oder blieb zumindest konstant. In den meisten Experimenten wurde mit dieser Methode eine Richtigkeit erzielt die vergleichbar ist mit der von Empfehlungssystemen, welche dem heutigen Stand der Technik entsprechen. Das Umrangierungsverfahren reduzierte die Dominanz von populären Artikeln in den Empfehlungslisten und ermöglicht es einen optimalen Ausgleich im Zielkonflikt der verschiedenen Leistungskriterien zu finden.

Abstract

We study graph vertex ranking algorithms for use in collaborative filtering-based recommender systems. In this paper we evaluate the performance of previously presented ranking algorithms in an off-line study with four different positive-only feedback datasets. Besides measuring the power to predict future user behavior (accuracy), we also consider four non-accuracy performance dimensions: intra-list diversity, item space or catalog coverage, personalization, and novelty/surprisal. We found that most recommendation lists of vertex ranking algorithms are dominated by high popularity items and give lower accuracy, coverage, personalization, and novelty/surprisal scores than lists from nearest-neighbor or latent factor model-based recommenders.

By applying a parametrized popularity-penalizing recommendation list re-ranking procedure to random walk vertex transition probability-based ranking algorithms (i.e., P^3 and P^5 [Cooper et al., 2014]) we observed a positive impact on coverage, personalization and novelty/surprisal. For small degrees of popularity penalization the recommender's accuracy improved or remained constant and reached in most experiments levels comparable to the state-of-the-art non-graph-based recommenders. The re-ranking procedure reduces the dominance of high popularity items in the recommendation list and allows to optimize the trade-off between accuracy and non-accuracy performance dimensions.

Contents

1

Introduction

Recommender systems are software systems that try to predict a user's preference for items and propose the most positively estimated items as suitable for consumption to the user. The output of real life implementations of recommender systems can be found in various domains. Typical examples are product recommendations in online stores or TV show recommendations of a video on demand service (VOD). Other areas of applications of recommenders comprise social networks, recommending other users as possible friends, or newspaper website, recommending a personalized and diverse mixture of breaking news. In all mentioned domains, users have to deal with a huge collection of items that is not readily comprehensible. With the means of recommender systems a personalized selection of items can be generated and thereby facilitate the users' decision-making process. This work investigates recommender systems based on graph theory. We describe the extension of a well-established recommender system framework with recommendation algorithms that analyze the network of user and item interactions. By comparison of the graph-based with other state-of-the-art recommendation approaches we demonstrate its potential and limitations.

Most research efforts in the field of graph-based recommenders focused on the improvement of the recommenders' predictive power (*accuracy*). However, it has been widely recognized [e.g. McNee et al., 2006; Ziegler et al., 2005] that users' satisfaction with recommendations is not only determined by the recommendations accuracy but also influenced by the diversity of the recommendations provided to a user. Consequentially, we consider in our study both performance characteristics. A recent paper showed that the most accurate graph-based recommendations are based on *vertex transition probabilities* of random walks of length three [Cooper et al., 2014]. The results of our experiments support this finding but also reveal that the diversity of recommendations from this and most other graph-based recommendation algorithms is low. In Chapter 2 we present different implementation strategies to calculate vertex transition probabilities and introduce a simple re-ranking procedure to increase recommendation diversity and accuracy applicable to the method proposed by Cooper et al. [2014]. In the same chapter we point out what we understand as diverse recommendations by introducing four different non-accuracy recommendation quality dimensions. The accuracy and diversity performance of graph-based recommenders is discussed in Chapter 3 based on off-line experiments with four positive-only feedback datasets. The first section of this chapter gives an overview over the different research directions in

the field of recommender systems. Subsequently we focus on the recent research efforts on graph-based recommender engines (Section 1.2) and then describe the motivation (1.3) for the study and our contributions (1.4).

1.1 Background

The problem of coming up with the most valuable set of recommended items for a given user has been tackled in multiple ways over the past decades. It can be formally described as finding the missing entries (zero entries) in the *user-item feedback matrix* F of size $|U|\times|I|$, where U is the set of all users and I the set of all items. If user u rated item i during the training phase $F_{u,i} \in \mathbb{R} \setminus \{0\}$ and $F_{u,i} = 0$ if u has not rated i. Hereafter we summarize the most important research directions in the field of recommender systems.

A first distinction of recommender system techniques can be drawn based on the input data used for the algorithms. *Content-based filtering* uses item meta data to analyze the characteristics of items that a user likes and searches for items with similar characteristics that are not yet known to the user. *Collaborative-filtering*, on the contrary, uses the preference feedback of users to search for users similar to a given user and recommends items that are preferred by the similar users yet unknown to the current user (collaborative filtering is often implemented item-centric by storing item similarities inferred from user feedback correlation in an item-item matrix). While the latter is nowadays the more popular technique that is applied in many real-world applications, both approaches are subject to ongoing research [Ricci et al., 2011]. More recently many recommender system implementations try to combine the two different input data types. Such systems are widely referred as *hybrid recommender systems* [Ricci et al., 2011].

Purely content-based approaches suffer from the problems of *limited content analysis* and *over-specialization* [Desrosiers and Karypis, 2011]. Limited content analysis means that it is often not possible to fully express an item's quality as well as a user's preference profile in terms of attributes. Over-specialization describes the problem that content-based algorithms tend to recommend items that are very similar to the items that a user rated positively in the past but fail to recommend items that are different but still interesting to the user.

Since content-based approaches do not require the feedback of other users to generate recommendations, items newly introduced into the system are as likely to be recommended as long-standing items. In collaborative filtering an item can only be evaluated as useful for a user once it received quality estimations from users considered as similar to the current user. This problem is the main disadvantage of collaborative filtering techniques and is known as the *cold-start* problem. The same problem arises for new users of the systems with none or only very few rated or liked items. The cold-start problem is a result of a challenge that most recommender systems face, namely a sparse user-item-feedback matrix. Simply expressed this describes the fact that the number of items that can be recommended to a particular user is for most users much

greater than the number of items for which a feedback from the user is available. Not only is the feedback matrix sparse but also the distribution of number of ratings among the items and users is very unbalanced. In most recommender systems a few users rate many items and most user rate only very few items, likewise a small fraction of items receive most ratings and a big number of items only a couple. Both dimensions of the feedback matrix, users and items, hence describe a *long-tail distribution* in the number of feedbacks.

An advantage of collaborative over content-based filtering is the absence of the need to discover meaningful user and/or item attributes. Also collaborative filtering is more likely to produce a more diverse result set thanks to the quality estimation of items by peers [Desrosiers and Karypis, 2011].

The collaborative filtering paradigm is widely implemented either with a *neighborhood* or a *model-based* approach [Koren and Bell, 2011]. Neighborhood-based methods are also known as *memory-based* methods and directly use the user-item ratings available from the training data for prediction. This can be done in two ways known as *user-based* or *item-based* recommendations [Desrosiers and Karypis, 2011]. *Graph-based* collaborative filtering is a neighborhood-based method. The training data are converted to a graph with *vertices* representing either the items, users or both, and *edges* the interaction between users, items or users and items. In order to generate recommendations the graph can be queried from a start vertex for similar vertices in its proximity for example by performing a *random walk*. Model-based methods, on the other hand, use the user-item ratings from the training set to learn a model of user item-interactions with factors representing latent characteristics of the users and items in the system [Desrosiers and Karypis, 2011]. Many popular model-based methods apply *matrix factorization*, for instance by *singular value decomposition*, to find the missing entries in the user-item feedback matrix.

This section gave a short introduction over the various types of recommender system developed in the past. We pointed out the differences between content-based and collaborative filtering and described some of the challenges and advantages of each approach. Due to the focus of this work, namely recommendation algorithms querying the user-item feedback graph, we emphasized on the developments in the field of neighborhood-based collaborative filtering. In the next section we delve into the literature of graph-based recommender systems.

1.2 Related Work

In this section we summarize how graph theory has been applied to the problem of recommendation generation in the past. Besides presenting the specific data models and algorithms of the recent research efforts, we also point out some important findings that can be used as guidelines for improvements. The names of the recommenders that we implemented in the framework extension and evaluated by their accuracy and diversity performance (see Chapter 3) are printed in bold face. We especially considered algorithms for the evaluation that are implementable by simulating random

graph walks.

Using a graph-theoretic approach for collaborative filtering in order to predict ratings was proposed for the first time in Aggarwal et al. [1999]. This paper introduces the concepts of *horting* and *predictability*. Horting describes the size of the intersection of the sets of the rated items of two users relative to their total size. Predictability, on the other hand, is a stronger property and requires that also the ratings of the items in the intersection are similar. The graph queried for rating prediction consists of vertices representing users and directed edges between users representing predictability. Predicting the rating of a new item i for user u is performed by using the shortest directed paths from u to other users that have rated i. We do not consider the data model and graph query algorithm of Aggarwal et al. [1999] in our evaluation of graph-based recommendation algorithms since we are focusing on positive-only feedback data and this does not allow the calculation of predictability between two users.

Huang et al. [2004] emphasize on the potential of applying an associative retrieval framework and related spreading activation algorithms to deal with the problem of sparse feedback data. Spreading activation algorithms essentially work by first activating a selected subset of vertices as starting vertices, and then iteratively activating the vertices that can be reached directly from the vertices that are already active, until a convergence criterion is met [Desrosiers and Karypis, 2011]. The authors propose three different types of spreading activation algorithms for a bipartite graph consisting of user and item vertices with undirected edges between vertices of the two sets that correspond to the user-item feedback. An empirical evaluation of the algorithm using data from an an online bookstore revealed comparable performance figures for the three different algorithms. We consider the algorithm referred in Huang et al. [2004] as **Branch-and-Bound (BnB)** in our evaluation. As a conclusion of their work the authors mentioned that the proposed spreading activation-based approaches alleviate the cold-start problem for new users. Furthermore, Huang et al. [2004] introduce the so-called three-hop algorithm as comparison baseline, which simply ranks the items by the number of paths of length three between the given user and an item. Items that already received feedback from the user, i.e. are known by the user, are not recommended. This algorithm is also considered in our evaluation and, in accordance with the notation used in Cooper et al. [2014] for the same algorithm, we refer to it as **3Path**. Besides the direct counting of distinct paths of length three between a user and an item, we also implemented an item-based 3Path version, called **IB-3Path**. IB-3Path precalculates the number of paths of length two between items and uses the resulting cache to speed up the graph search for the number of paths to items in three edge distance from a given user.

The first researchers who introduced random walks on graphs interpreted as Markov chains for collaborative filtering were Fouss et al. [2005]. The vertices in the graph represent, very similar as in the data model of Huang et al. [2004], users and items. Edges stand for the feedback of a user for a particular item and are undirected and weighted, e.g., by the number of times the user consumed an item or a number expressing the rating of the item by the user. Fouss et al. [2005] associate a state of the Markov chain with each vertex of the graph. The transition probabilities to adjacent

vertices, respectively to a different state of the Markov chain are proportional to the weights of the edges connecting the vertices. Hence the transition probabilities depend only on the current state and not on the past ones (first-order Markov chain). Fouss et al. [2005] then introduce a set of quantities to rank vertices by their similarity to a start vertex using the Markov chain. In this work we consider the ranking algorithms **Average Commute Time (CT)** and **Average first-passage time (OW)**. Both ranking algorithms showed good performance and are implementable by simulating random walks. Average first passage time measures the number of steps a random walk needs on average to reach a particular item from a given user. Average commute time extends this concept by defining a random walk as path between start user to target item and back to start user. Both metrics consider an item as a good recommendation if the average length of the random walk is short. Furthermore, we also consider the best performing scoring algorithm proposed in Fouss et al. [2005], using the vertex similarity measure provided from the **Moore-Penrose pseudoinverse of the Laplacian matrix ($\mathbf{L}^+$)**[1] of the graph.

The random walk based graph query approach was adopted by Gori and Pucci [2007] for a vertex scoring algorithm named **ItemRank (IR)**. In contrast to the data model used by Fouss et al. [2005] ItemRank runs on a graph consisting only of one type of vertices representing items. The authors call this data structure a correlation graph since its weighted and directed edges model the correlation between items based on the number of users that gave feedback on both items. The basic idea of the ItemRank algorithm is to spread user preferences through the correlation graph starting from the items a user likes. By *propagation* of the initial preferences to adjacent vertices the algorithm discovers similar items, while *attenuation* of the preferences on the other side assures that more distant items are considered as less similar. The two properties propagation and attenuation are satisfied by the *PageRank* [Page et al., 1998] algorithm. Gori and Pucci [2007] use a biased version of PageRank very similar to the *topic-sensitive PageRank* of Haveliwala [2002] to rank items for a given user. ItemRank was evaluated on the MovieLens[2] dataset and compared to the algorithms, among others, proposed by Fouss et al. [2005]. The experiments show that ItemRank performs better than the benchmark algorithms in terms of prediction accuracy while the algorithm is less complex with respect to memory usage and computational cost. The reduced complexity is attributed to the reduced size of the correlation graph compared to a bipartite graph consisting of user and item vertices as used by Fouss et al. [2005] and Huang et al. [2004].

Cooper et al. [2014] contribute three new methods referred as $\mathbf{P^3}$, $\mathbf{P^5}$ and $\mathbf{P^3_\alpha}$. The authors distinguish in their work two approaches to order vertices based on the vertex transition probabilities when performing short random walks over a undirected bipartite user-item graph: (i) calculating the exact values of the transition probability matrix using matrix algebra or (ii) estimating the matrix entries by simulating random

[1]The Laplacian matrix L of a graph is defined as $L = D - A$ where D denotes the diagonal matrix of the vertex degrees and A the graphs adjacency matrix.

[2]The MovieLens dataset was constructed from an online movie recommendation engine and is available in three different sizes from *grouplens.org/ datasets/ movielens/*

walks (see Section 2.1 for a complete introduction of both methods). The probabilities obtained with approach (ii) converge to the result of (i) with increasing number of random walks. In the same paper it is shown that the time- and memory-efficiency of direct simulation of random walks allow application of vertex transition probability-based methods to large datasets with only limited negative impact on accuracy. This finding motivates our decision to focus our evaluation on vertex ranking algorithms that are implementable by the simulation of graph walks. P^3 and P^5 perform a random walk of fixed length 3 and 5, respectively, starting from a given user vertex. Due to the fact that the graph is bipartite all destination vertices represent items. Cooper et al. [2014] report a better accuracy for the P^3 and P^3_α algorithm than for the methods proposed in Fouss et al. [2005] and Gori and Pucci [2007] in a evaluation on the MovieLens data. P^3 and P^3_α also outperform the computationally more demanding P^5 method. P^3_α is a parametrized version of P^3 that offers further improvements over P^3 in accuracy for empirically determined values of the parameter α. In the remainder of this work we refer to the α controlled optimization of transition probabilities with *transition probability tuning*. How to simulate P^3_α with random walks is not clearly stated in the work.

Researchers at Google analyzed the user-video graph of YouTube [Baluja et al., 2008] to provide personalized video recommendations and summarized three rules for a good recommendation, i.e., the user u will have a high preference on the item i if: (i) u and i have a short path between them; (ii) u and i have several paths between them and these paths are not very long; (iii) u and i have paths that avoid high-degree vertices. While the validity of the first two rules are obvious the third can be explained by the fact that it is quite likely to find a short path through one or multiple high degree vertices between any two vertices in the graph. Hence even though such a path is short, it does not mean that e.g. two items are similar. Baluja et al. [2008] grasp recommendation generation as a label propagation task in a graph and present a new group of algorithms, termed *Adsorption*, that computes for each vertex a label distribution. Adsorption can be implemented as a random walk. An interesting extension to the classical random walk model is the introduction of an *abandonment probability* that causes the walk to stop without contributing to the final label distribution. This parameter can for instance be used to prevent random walks through high degree vertices. We decided against considering the Adsorption algorithms in our evaluation since its random walk implementation is highly inefficient according to the authors.

Other research efforts in the field of graph-based recommendations try to improve recommendation quality by incorporating meta data or contextual information in the data model [Bogers, 2010; Jamali and Ester, 2009; Lee et al., 2012; Xiang et al., 2010]. Xiang et al. [2010], for example, show how the temporal context of user feedbacks can increase prediction accuracy. The authors distinguish between short- and long-term preferences of users by assigning timestamped user feedbacks to sessions which are then mapped into a weighted user-item feedback graph as links between items and session vertices. The vertex ranking is obtained from this graph by summing the weights of all paths of length three from the user vertex to the item vertex and the weights of all

paths starting from the current session vertex to the item vertex.

In the recommender system research community it has been increasingly noted that it is not sufficient to have accuracy as the sole criteria to measure recommendation quality [Adomavicius and Kwon, 2011]. Other aspects that influence the satisfaction of a user with a set of recommendations are for instance diversity, novelty, serendipity, confidence, or trust [Adomavicius and Kwon, 2011; Herlocker et al., 2004; McNee et al., 2006; Shani and Gunawardana, 2011]. This change in paradigm led Adomavicius and Kwon [2011] in their development of a graph-based recommender that maximizes aggregated diversity, which is very similar to what we call *item-space coverage*, based on maximum flow or maximum bipartite matching computations.

Zhou et al. [2010] study the performance of vertex ranking algorithms for bipartite user-item networks in two different non-accuracy dimensions: *Personalization* and *Surprisal/Novelty* (see Section 2.3 and Subsection 3.1.3). The authors report a simultaneous improve in accuracy and non-accuracy performance for a parametrized algorithm that combines the item rankings of an accurate vertex ranking algorithm with a ranking algorithm that is less accurate but gives better personalization and surprisal scores. The former is a random walk vertex transition probability-based ranking algorithm very similar to P^3 [Cooper et al., 2014] that favors highly connected items. The latter represents a discrete analogy of a heat diffusion process and favors vertices with few items.

In this section we described recent research efforts and pointed out important findings in the field of graph-based recommender algorithms. We focused on the description of algorithms that are based on the collaborative filtering paradigm but also mentioned some papers that present hybrid graph-based recommenders. We conclude the section by referencing graph-based methods that improve non-accuracy recommendation quality.

1.3 Motivation

In this section we first explain our interest in the application of vertex ranking algorithms for recommendation generation. We then point out the importance of non-accuracy performance to assess the quality of a recommender system.

Traditional neighborhood-based collaborative filtering techniques, mentioned in Section 1.1, suffer from the flaw of *limited coverage* according to Desrosiers and Karypis [2011]. Limited coverage refers to the problem that two users are only considered neighbors and can therefore be used to infer recommendations, if they have rated common items. This requirement is needlessly restrictive, as users without commonly rated items may still have similar preferences. Data sparsity aggravates the problem of limited coverage. Desrosiers and Karypis [2011] mention graph-based recommenders as one method to overcome limited coverage, thanks to the underlying data model that makes transitive relationship information accessible to queries.

Recommender systems based on walks over a graph are also attributed with high flexibility in terms of incorporating contextual information (e.g. location, time or

season) or user (e.g. age, gender or occupation) or item (e.g. tags, genre, year) meta data [Lee et al., 2012]. Extending the data model with the item meta data genre of movies can be for instance achieved by linking movie vertices of a given genre to a dedicated vertex representing that genre. With these newly introduced connectivities a graph walk can follow additional paths and thereby generate a different set of recommendations compared to pure graph-based collaborative filtering.

A recommender system that generates recommendations based on graph walks over a user-item-feedback graph does not require an computationally expansive training phase: the required training time increases linear with the number of feedbacks, which is advantageous when compared to many state of the art recommenders. Also incorporating new users, items, or ratings in the data model is computationally efficient since it consists simply of expanding the graph with additional graph elements. While the training of a graph-based recommender might be comparably cheap, the generation of recommendations requires the execution of computationally costly graph queries. This means that graph-based recommender systems require the majority of the totally required processing power during recommendation and not model generation time.

Ahn et al. [2013] show how graph-based recommendations can be visualized by using the well known and easily comprehensible graphical representation of graphs consisting of points for vertices connected by lines representing edges. By means of an intuitive graphical depiction, a set of generated recommendations can be explained and justified to an audience that is not familiar with the applied recommender algorithm. This can be advantageous when it comes to implementing and maintaining a recommender system in a production environment. Furthermore, it is thinkable that the possibility to intuitively depict graph-based recommendation makes it feasible to engage the user of a recommender system in the process of recommendation generation.

Already several years ago, leading recommender system researchers emphasized that focusing on prediction accuracy as the only criteria to assess the quality of a recommender is insufficient [McNee et al., 2006]: they summarize that the narrow focus on accuracy has been misguided and detrimental to the field. In the same paper the authors mention the research efforts presented in Ziegler et al. [2005] as a possible approach to overcome this shortcoming. Ziegler et al. [2005] introduced a recommendation list re-ranking procedure that improves the lists diversity measured by the variety of item attributes occurring in the list. The re-ranking procedure reduced the accuracy of the recommendation lists. However, the user satisfaction increased for slightly diversified lists and dropped again below the score of the undiversified list for strongly diversified lists. From this result we can conclude that a recommender system that maximizes user satisfaction has to find the optimal trade-off between accuracy and diversity.

Another non-accuracy performance criteria that is considered as a desired property of a recommender system is the capability to include items of low popularity, often referred as *long-tail items*, in the recommendation lists and thereby suggesting items users would not readily discover by themselves [Adomavicius and Kwon, 2012; Herlocker et al., 2004; Shani et al., 2008; Zhou et al., 2010]. Zhang et al. [2012] report an increase in user satisfaction for recommendation lists that include more items of

low popularity. Moreover Adomavicius and Kwon [2012] point out that from an item provider's point of view it can also be beneficial to increase the diversity of the aggregated recommendation lists, which is not necessarily high if the diversity of single recommendation lists is high (see also Section 2.2). Despite the widely recognized importance of non-accuracy metrics to assess a recommender systems performance, most studies of graph-based recommenders consider accuracy only.

1.4 Contributions

The main contributions of this work are (i) the analysis of the performance characteristics of graph-based recommenders with a multidimensional approach (see Section 2.3) and (ii) a vertex re-ranking procedure to increase the number of long-tail items in a recommendation list (see Section 2.2). We assess the performance of graph-based recommenders with a set of accuracy, non-accuracy and relevance-aware non-accuracy metrics (see Subsection 3.1.3). The pursued evaluation includes also neighborhood and latent factor model-based recommenders and thereby sheds light on the accuracy and non-accuracy performance potential of vertex ranking algorithm compared to state-of-the-art recommendation approaches. Using the findings of our evaluation we develop a popularity-dependent parametrized re-ranking procedure that can improve accuracy and non-accuracy performance of vertex transition probability-based vertex ranking algorithms. In our experiments we found a significantly (Wilcoxon $p<0.0001$, see Appendix A.3) better accuracy and low-popularity item recommendation performance for all datasets with the re-ranking procedure applied to P^3 [Cooper et al., 2014] than for P^3_α, a parametrized version of P^3 that maximizes accuracy. The parameter of the proposed re-ranking procedure can be used to find a good trade-off between accuracy and non-accuracy performance.

We focus in our work on positive-only feedback data (sometimes also referred as implicit feedback): for any non-zero entry in the binary user-item feedback matrix F we assume that the corresponding user u likes the corresponding item i, i.e. $F_{u,i} = 1$. Rating values, even if available from a dataset, are neglected. This limitation of our work is justified by the fact that in many domains where recommendation generation is desirable, no explicit ratings are available but positive-only feedbacks can be inferred from the service usage statistics. For positive-only feedback data the recommendation generation task translates into ranking the items for a given user according to descending user preference.

We study graph-based recommenders because these algorithms allow to overcome the problem of limited coverage and have interesting computational properties. Furthermore, this approach facilitates creating hybrid recommender systems by combining meta and feedback data in a uniform data model. Our interest in non-accuracy performance characteristics of recommenders is well-grounded by several research efforts in this direction that emphasize the importance of diversity, novelty, or aggregated diversity for a successful recommender. However, non-accuracy performance of graph-based recommenders have not yet been systematically studied. With the present work

we intend to fill this gap in the research on recommender systems.

2

Diverse Graph-based Recommendations

So far, we stated the motivation for our interest in studying vertex ranking algorithms for the use in collaborative filtering recommender systems and presented previously undertaken research efforts in this field. In this work we focus in particular on vertex ranking algorithms based on vertex transition probabilities after repeated short random graph walks as proposed in Cooper et al. [2014]. In the first section of this chapter we show different approaches to calculate vertex transition probabilities. Subsequently (Section 2.2), we introduce a recommendation re-ranking procedure that we call *popularity normalization* to include long-tail items in a recommendation set. After describing our conception of important recommendation diversity dimensions (Section 2.3), we conclude the chapter with a description of the software setup that we used for the analysis of vertex ranking algorithms (Section 2.4).

2.1 Recommendations based on Vertex Transition Probabilities

The vertex ranking algorithms considered in Cooper et al. [2014] are based on random walks over the undirected bipartite graph constructed from the users' feedback on items (*user-item-feedback graph*). The vertices V of the user-item-feedback graph $G = (V, E)$ represent the union of the two entity sets users U and items I ($V = U \cup I$). The graph's edge set is $E \subseteq U \times I$ and for each edge $e \in E$ holds $e = \{u, i\}$, where $u \in U$ and $i \in I$. This means that the graph contains only edges between a user and an item vertex and hence the graph is bipartite. An edge exists in the graph between u and i if the corresponding entry in the user-item feedback matrix F is non-zero ($F_{u,i} \neq 0$). Hence each user is connected to all items that the user rated and an item is connected to all users who rated the item. All edges in the graph are unweighted and no parallel edges exist. However, edge weights or parallel edges, for instance based on rating values or the number of interactions, could be used for a more accurate representation of the users preference profile. As in Cooper et al. [2014] we do not consider this extension in the presented work.

The vertex ranking algorithm P^{s}, as well as the more general parametrized method $\mathrm{P}^{\mathrm{s}}_{\alpha}$, proposed in Cooper et al. [2014] are based on the distribution of random walks

after s steps, which is the s-th power of the transition probabilities matrix P. The recommendation lists of these methods are obtained by ordering the items according to descending transition probabilities of the corresponding vertices. If we want to obtain a vertex ranking useful as item recommendation list for u we are interested in the random walk distribution after an odd number of steps greater than one, i.e. $s \in \{3, 5, \dots\}$. We investigated three approaches to obtain P^s: direct calculation with matrix algebra, simulation of random walks, and probability tree traversals.

The adjacency of the vertices of the user-item feedback graph G can be represented in a square matrix of size $|V|$ called adjacency matrix A. Since all edges in G are unweighted and undirected the entry $a_{i,j}$ in A is equal $a_{j,j}$ (A is symmetric) and 1 for two connected vertices i and j and 0 otherwise. Furthermore we define the degree matrix D of a graph. D is a diagonal square matrix of size $|V|$ with diagonal elements $d_{i,i} = \sum_{j=1}^{|V|} a_{i,j}$. Since D is a diagonal matrix and we assume that all diagonal entries are non-zero (i.e., no unconnected vertices) its inverse D^{-1} is given by $(d_{i,i}^{-1})$, and hence cheap to compute. Using the inverse degree matrix D^{-1} and A we obtain the transition probability matrix P with a simple matrix multiplication

$$P = D^{-1}A \tag{2.1}$$

We can generalize P to P_α by raising each entry of P to the power of parameter α ($\alpha \in \mathbb{R}$), thus $P = P_{\alpha=1}$. The ranking determining matrix P_α^s is obtained from the s-th power of P_α. Note that the entries of the transition probability matrix after s steps with $\alpha = 0$ is equal to the normalized number of paths of length s, hence the algorithm $\mathrm{P}^3_{\alpha=0}$ is equivalent to 3Path. We refer to results obtained with matrix algebra calculation with the postfix *(M)* (e.g., 'P^3_α (M)' denotes the P^3_α method calculated by matrix algebra operations).

Beside direct calculation of P^s with matrix algebra, we also estimated the transition probability distribution for a given user u by simulating random walks (see Algorithm 1). Cooper et al. [2014] showed that the distribution obtained with a simulation of random walks converges to the distribution from matrix algebra calculation for an increasing number of simulated walks. At the start of the simulation algorithm the current vertex v_c is the vertex of the start user. Next v_c is assigned s-times a randomly selected neighboring vertex of v_c. Subsequently the value mapped to v_c, the terminal vertex of the walk, in an associative array is increased by one (with 0 as the default value of the associative array). This procedure is repeated until the ordering of the vertices according to the values in the associative array remains stable. After normalizing by the number of simulated random walks, the values in the associative array returned from Algorithm 1 approximate the entries in P^s for u. In our experiments we used a simple heuristic as convergence criterion: the vertex ranking was assumed as stable if the ordering of the top n items did not change for r iterations. If this criterion was not reached after $2'000'000$ restarts the simulation was aborted and we used the resulting vertex ranking for recommendation generation. We set $n = 20$ and $r = 50'000$, except for the MovieLens-S dataset (see Section 3.1 for dataset description) where $n = 5$ and $r = 20'000$. Note that Algorithm 1 estimates

random walk distributions according to P^s. Simulating the underlying random walk distribution of the parametrized method P^3_α is not possible with Algorithm 1. We refer to results based on simulations of random walks with the postfix *(S)*.

Algorithm 1 Estimating vertex transition probabilities by simulating random walks with restarts.

Require: v_u is the vertex representing user u and s the length of a single random walk

```
1: function EstimateP(v_u, s)
2:     m ← an associative array with default value 0
3:     while !Converged(m) do
4:         v_c ← v_u
5:         for 1 to s do
6:             v_c ← GetRandomNeighbor(v_c)
7:         end for
8:         m[v_c] ← m[v_c] + 1
9:     end while
10:    return Normalize(m)
11: end function
```

The third approach, denoted by the postfix *(T)*, calculates the random walk distribution based on the idea of traversing a probability tree starting from user u's vertex (v_u) up to depth s (see Algorithm 2). Unlike to the simulation approach, probability tree traversal allows to calculate entries of P^3_α for any value of α and not only for $\alpha = 1$. The transition probabilities for u are obtained by following all paths of length s starting from v_u. In the first step the total transition probability of 1 is divided by the number of neighbors of v_u. This step is repeated for each neighbor using the fraction of the probability until s edges were traversed. At length s of the walk the remaining probability fraction is raised to the power of α and added to the value (initially 0) stored in an associative array for the current vertex. The associative array returned by Algorithm 2 contains the transition probabilities from v_u to any vertex after a random walk of length s. This approach for random walk distribution calculation has the advantage over the simulation with random walks that it gives exact results and allows transition probability tuning with α. Compared to the calculation of P^s with matrix algebra this approach avoids matrix manipulations that become quickly infeasible with increasing dataset size since the matrices do not fit into computer memory. Calculating the random walk distribution by traversing a probability tree can also be implemented in the graph element centric computation model of the Signal/Collect framework [Stutz et al., 2010]. In Appendix A.4 we show the necessary implementations using Signal/Collect to calculate the transition probabilities of an example graph. The proposed implementation uses the synchronous execution mode of Signal/Collect, which is closely related to the Bulk Synchronous Parallel (BSP) computing model and Pregel [Malewicz et al., 2010].

Algorithm 2 Calculation of vertex transition probabilities by probability tree traversal.

Require: v_u is the vertex representing user u, s the length of a single random walk, and α the transition probability tuning parameter

```
1: function CalculateP(v_u, s, α)
2:     m ← an associative array with default value 0
3:     TraverseTree(v_u, m, 0, 1, s, α)
4:     return m
5: end function

6: function TraverseTree(v_c, m, depth, p, s, α)
7:     if depth < s then
8:         neighbors ← GetAllNeighbors(v_c)
9:         p ← p / |neighbors|
10:        for each v_n in neighbors do
11:            TraverseTree(v_n, m, depth + 1, p, s, α)
12:        end for
13:    else
14:        m[v_c] ← m[v_c] + p^α
15:    end if
16: end function
```

2.2 Popularity-normalized Graph-based Recommendations

Our analysis of different type of graph-based recommendation algorithms (see Section 3.2) showed that the resulting item ranking of the most accurate methods is strongly influenced by the popularity of the items, i.e., the most popular items appear for most users at the top of the recommendation list. We assume that such a ranking is undesirable for multiple reasons. (i) It is likely that a user is already aware of the most popular items and hence the recommender system does not help users to discover unknown items. (ii) A dominating influence of item popularity makes it less likely for novel items to get recommended even if they match a user's preference profile and thereby aggravates the item cold-start problem. (iii) We believe that in order to engage a user with recommended items, the proposals should convey a feeling of personalization to the user. (iv) If we think of a recommender system as a means for content providers to advertise items, the capability to recommend low popularity items (which in turn increases aggregated diversity) to appropriate users is probably a desirable performance attribute. Adomavicius and Kwon [2011] ground the desirability of increased aggregated diversity with a nice example: a Video-on-Demand (VoD) provider (e.g., Netflix) should encourage users to rent more long-tail movies because these items are less costly to license and acquire from distributors than new releases or extremely popular movies of big studios.

For these reasons we investigated algorithmic extensions to existing graph-based recommendation methods that introduce item popularity as ranking influencing quantity. We refer to these extensions as *popularity normalization*. For the P^s type of algorithms (see Section 2.1), item ranking is based on the transition probability matrix P of a random walk raised to the power s. Due to the randomness of the walk with uniform edge selection probabilities and the high vertex degree of popular items, it is likely that the walk ends at a popular item. To compensate for this effect we propose to rank items by a popularity normalized quantity (Q) of type

$$Q_{u,i} = P^s_{u,i} * f(d_{i,i}) \tag{2.2}$$

where $P^s_{u,i}$ is the transition probability after s steps from user u to item i and $f(d_{i,i})$ a function depending on the popularity of i in the training data. $d_{i,i}$ is the entry for vertex i (here representing an item) in the degree matrix of the user-item-feedback graph. In principle we can replace the transition probability $P^s_{u,i}$ in Equation 2.2 with any other property that describes the relationship between two graph vertices, e.g., the average hit or commute time between a start and target vertex or the entry for u and i in the pseudoinverse Laplacian matrix $L^+_{u,i}$. Since transition probability-based graph recommendation methods showed the best accuracy performance in our preliminary analysis (see Section 3.2) we examined the effect of popularity normalization only for these methods.

We experimented with the following two parametrized normalization functions:

$$f_a(d_{i,i}) = \frac{1}{d^\beta_{i,i}} \tag{2.3}$$

and

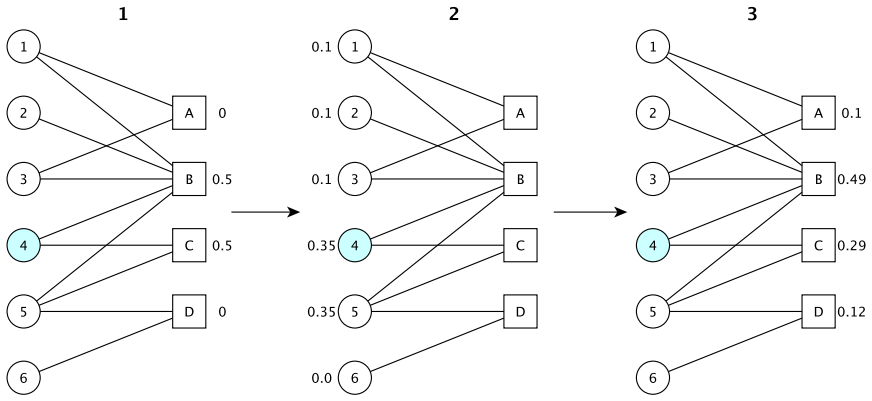
$$f_b(d_{i,i}) = \frac{1}{rank(d_{i,i})^\beta} \tag{2.4}$$

where $\beta \in \mathbb{R}$ is a tuning parameter for the degree of popularity normalization and $rank(d_{i,i})$ calculated with the following algorithm:

1. $l = \langle d_{1,1}, ..., d_{n,n} \rangle$ is the list of the item degrees ordered ascending
2. Let k ($k \in 1, 2, ..., n$) be the index of $d_{i,i}$ in l. Then $rank(d_{i,i}) = rank(k) = 1 + \sum_{j=2}^{k} g(l_{j-1}, l_j)$ where

$$g(l_a, l_b) = \begin{cases} 1, & if \quad l_a < l_b \\ 0, & otherwise \end{cases} \tag{2.5}$$

If we choose $\beta = 0$, then $f_a(d_{i,i}) = f_b(d_{i,i}) = 1$ and thus the item ranking is fully determined by the transition probability matrix ($Q_{u,i} = P^s_{u,i}$). A positive β value has the effect of reducing the transition probability to low degree vertices to a smaller extent than to high degree vertices and hence it becomes more probable for unpopular items to appear at the top places of a recommendation list. If we choose a negative



Item	P^3_α with α						AN-P^3 with β				
	-1.0	0.0	1.0	2.0	3.0	4.0	-0.5	0.0	0.5	1.0	1.5
A	0.147	0.133	0.100	0.062	0.033	0.016	0.078	0.100	0.122	0.142	0.158
B	0.441	0.467	0.492	0.512	0.526	0.533	0.605	0.492	0.380	0.279	0.197
C	0.257	0.267	0.292	0.326	0.359	0.387	0.227	0.292	0.356	0.414	0.461
D	0.154	0.133	0.117	0.100	0.082	0.064	0.091	0.117	0.142	0.165	0.184

Figure 2.1: Random walk vertex transition probabilities after 1, 2, and 3 steps (Figures 1-3) and resulting ranking weights (Table) of a single user in a toy graph. Circles represent user vertices (1-6) and squares represent item vertices (A-D). The target user (4) is colored in cyan.
Vertex transition probability tuning (P^3_α) attenuates the spread of user preference and increases the difference of the ranking weights of the two items unknown to the target user (items A and D) for increasing values of $\alpha > 0$. The weight of item D decreases to a smaller extent than of item A because the two 3 edge distance paths from user 4 to item D pass through vertices of lower degree than the two paths leading from user 4 to item A. The difference in the ranking weights of the two unknown items also increases for popularity normalization (AN-P^3) with increasing values of β. However, more pronounced is the redistribution of the user preference from high degree to low degree vertices. Note that P^3_α with $\alpha = 1$ and AN-P^3 with $\beta = 0$ (as well as P^3) are the same algorithm.

β the popular items are even more likely listed at the top of a recommendation list than with a pure transition probability-based ranking. In Figure 2.1 we illustrate on a toy graph the influence of popularity normalization and transition probability tuning (see Sections 1.2 and 2.1) on the item rank determining weights. In the remainder of this work we refer to recommendation algorithms using the absolute popularity of an item for normalization (f_a, Equation 2.3) with the prefix AN- and to algorithms with popularity rank-based normalization (f_b, Equation 2.4) with the prefix RN-.

2.3 Dimensions of Recommendation Diversity

In this work we consider four non-accuracy performance dimensions for evaluating recommendation quality: *intra-list diversity*, *item-space or catalog coverage*, *personalization*, and *novelty/surprisal*. In contrast to *accuracy*, denoting an algorithm's predictive power, we use the term *diversity* when we refer to any of the non-accuracy

dimensions. See subsection 3.1.3 for a formal description of the metrics used to measure a recommender's non-accuracy performance.

Intra-list diversity measures the quality of a set of recommended items in terms of the diversity of these items among each other. In this context, the diversity of two items is determined by the distance between the items in the space of item attributes. More diverse items are further apart in the space. For instance in the case of movie recommendations an item attribute could describe the genre of the movie. If a set of recommendations consists only of movies with the genre attribute value "Western" the intra-list diversity score would be low.

Item space or catalog coverage, sometimes also called aggregated diversity, is an indicator for the uniformity of the abundance of the recommendable items in the aggregated recommendation sets. A recommender achieves perfect catalog coverage if each item appears with equal probability in a set of recommendations, as it is the case for the random recommender. In the remainder of this text we refer to this diversity dimension simply with coverage.

A random selection of recommended items gives also the highest possible score in personalization, which evaluates how similar a generated set of recommendations is when compared to all other sets. A highly personalized set of recommendations contains only few items that appear also in other sets. We expect that the recommendation performance in the coverage and personalization dimensions are positively correlated: a recommender with a low coverage recommends the same items multiple times and therefore the difference among recommendation sets is likely to be smaller. This is equivalent to a decrease in personalization.

The novelty/surprisal diversity dimension evaluates a single recommendation set based on the popularity of the items in the set. Since a user is less likely to be aware of items with low popularity, the novelty/surprisal score of a recommendation set is high if it consists of low popularity items. Consequently, recommending items based on a ranking by descending item popularity gives the worst novelty/surprisal scores. If a recommender does not only recommend the very popular items, the aggregated recommendation sets comprise a greater number of different items, i.e., the coverage of the recommender increases. For this reason we also expect a positive correlation between novelty/surprisal and coverage. In the remainder of this text we refer to the novelty/surprisal diversity dimension simply with surprisal.

2.4 System Setup

We extended the Java port of the MyMediaLite recommender system framework[1] with two additions: (i) a set of metrics (see Section 3.1) measuring recommendation performance according to the diversity dimensions introduced in Section 2.3 and (ii), a component implementing the graph-based recommender algorithms introduced in Section 1.2, which depends on the MyMediaLite framework. The original MyMedi-

[1] MyMediaLite port from C# to Java - *github.com/jcnewell/MyMediaLiteJava*

aLite library[2] is described by its authors as a lightweight, multi-purpose library of recommender system algorithms that addresses the two most common scenarios in collaborative filtering: rating prediction and item prediction from positive-only feedback [Gantner et al., 2011]. It contains a wide variety of memory and model based collaborative filtering recommender algorithms, dedicated evaluation metrics for both recommendation scenarios and allows the incorporation of contextual or meta data through a shared data input and output mechanism. The Java port of MyMediaLite is almost as feature rich as the original version and copes well with the Java-based graph processing technology stack that we use for the framework extension. The graph-based recommender algorithms share a common superclass, inheriting from the MyMediaLite `ItemRecommender` class, that constructs the graph data structure during the training phase based on the boolean user-item feedback matrix provided by the framework. The graph is a `TinkerGraph` object implementing the Blueprints[3] `Graph` interface and queryable with a domain specific language called Gremlin[4]. Various

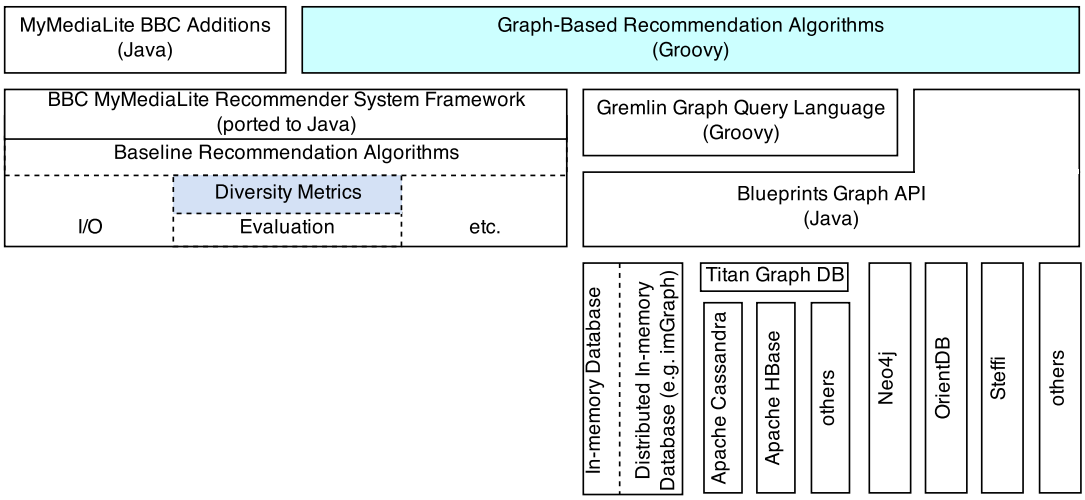


Figure 2.2: Enabling graph-based recommendations with the MyMediaLite framework. The colored components of the setup were developed in the context of this work.

types of storage implementations, both persistent and in-memory, are available for the Blueprints graph API. The back-end used in our experiments is memory-based and hence the size of the processable graph is limited by the memory available on the computer. However, it is thinkable to apply graph walk-based recommendation algorithms to larger datasets by using a more scalable Blueprints enabled storage solution. The dependences between the system components together with the underlying technologies are presented in Figure 2.2.

[2]MyMediLite recommender system library - *github.com/zenogantner/MyMediaLite*
[3]Blueprints generic graph API - *github.com/tinkerpop/blueprints*
[4]Gremlin graph traversal language - *github.com/tinkerpop/gremlin*

3

Evaluation

In the previous chapter and Section 1.2, we introduced different approaches for graph-based recommendation algorithms. In the following sections we will present and discuss the algorithms accuracy and diversity performance characteristics based on the results of off-line experiments with four different datasets. We start by outlining the evaluation methodology (Section 3.1), including a detailed explanation of the used diversity metrics (3.1.3), followed by an assessment of the performance of graph-based recommenders (3.2) when compared to a set of baseline recommenders. Subsequently we discuss the achieved improvements in accuracy and diversity with popularity normalization (3.3) and compare it to the potential of transition probability tuning (3.4) as proposed in Cooper et al. [2014]. We show in Section 3.5 how popularity normalized graph-based recommenders perform compared to three different baseline methods, again considering accuracy and the four different diversity dimensions. To exemplify the effect of popularity normalization we include recommendation lists generated with increasing degree of popularity normalization for a sample user in Section 3.6. The chapter is concluded by presenting results of exploratory experiments combining popularity normalization and transition probability tuning (3.7).

Key findings of the evaluation are summarized in margin notes

3.1 Methodology

This section starts with an overview over the baseline recommenders considered in the evaluation followed by a description of the dataset characteristics and the used performance metrics.

3.1.1 Baseline Recommenders

We measured the performance of the graph-based recommendation algorithms introduced in Section 1.2 in terms of prediction power of future user behavior, hereafter referred as accuracy, and different diversity metrics. The same measurements were performed for weighted (**WI-kNN**) and unweighted (**I-kNN**) k-nearest neighbor item-based collaborative filtering recommenders using cosine distance as item similarity measure. Both algorithms are readily available in the MyMediaLite library. We experimented with different neighborhood sizes varying from 10 to 1'600. We also

Name	Ratings	Users	Items	Min. User Degree	Max. User Degree	Min. Item Degree	Max. Item Degree	Median User Degree	Median Item Degree	Avg. User degree	Avg. Item Degree	Sparsity	Graph Diameter
MovieLens-S - total	100000	943	1682	20	737	1	583	65.0	27.0	1.1×10^2	5.9×10^1	6.3×10^{-2}	5
MovieLens-M - total	1000047	6038	3706	20	2314	1	3426	96.0	123.0	1.7×10^2	2.7×10^2	4.5×10^{-2}	
MovieLens-M - train	700047	6038	3706	10	1576	1	2407	67.0	86.0	1.2×10^2	1.9×10^2	3.1×10^{-2}	6
MovieLens-M - test	300000	6038	3513	1	738	1	1019	28.0	42.0	5.0×10^1	8.5×10^1	1.4×10^{-2}	
iPlayer - total	4703471	655846	808	1	170	5	147987	6.0	1867.5	7.2	5.8×10^3	8.9×10^{-3}	
iPlayer - train	4691493	655846	808	1	170	5	147674	6.0	1860.5	7.2	5.8×10^3	8.9×10^{-3}	6
iPlayer - test	14616	5000	623	1	30	1	714	2.0	8.0	2.9	2.3×10^1	4.7×10^{-3}	
BookCrossing - total	369195	4052	18280	20	5399	1	872	42.0	12.0	9.1×10^1	2.0×10^1	5.0×10^{-3}	
BookCrossing - train	258436	4052	18280	15	3844	1	616	30.0	8.0	6.4×10^1	1.4×10^1	3.5×10^{-3}	7
BookCrossing - test	110759	4050	16768	1	1555	1	256	13.0	4.0	2.7×10^1	6.6	1.6×10^{-3}	

Table 3.1: Datasets properties. Graph diameters are estimates obtained with the built-in function `GraphDiameter` and option `PseudoDiameter` of *Wolfram Mathematica 10*®.

considered a recommender based on a latent factor model obtained with matrix factorization, called **BPRMF** [Rendle et al., 2009] (also contained in the MyMediaLite library). We maximized the accuracy of BPRMF by varying the number of latent factors in the range between 10 and 200. See Appendix A.1 for the complete sets of parameters used for each recommender. In addition to these three personalized baseline algorithms we included two non-personalized recommendation strategies in our analysis: global item popularity (**MostPop**) and random item ranking (**Random**). To facilitate comparison we also calculated all metrics for the perfect recommender (**Perfect**) which places all test items of a user at the top of the recommendation list (with random ordering among the test items). All accuracy figures presented in this work are relative to the performance of the Perfect recommender. Due to the varying computational time and space complexity of the different graph-based and baseline recommendation algorithms, we could not measure the performance of each algorithm for all datasets. In particularly it was not possible to perform experiments with the largest dataset (iPlayer, see Section 3.1.2) for graph-based recommenders that use matrix algebra operations (P^3 (M), P^3_α (M), P^5 (M), P^5_α (M), and L^+ (M)) because of the large size of the graph's degree and adjacency matrices. Table A.1 in Appendix A.2 summarizes the experiments performed for the evaluation.

3.1.2 Datasets

We used four different datasets for our experiments: the 100K MovieLens dataset (hereafter referred to as MovieLens-S), the 1M MovieLens dataset (MovieLens-M), the Book Crossing dataset (BookCrossing)[Ziegler et al., 2005], and a dataset obtained from the iPlayer called VoD system of the British Broadcasting Corporation (iPlayer). Since this work addresses recommendation generation based on implicit user feedback we neglected the rating values available in the MovieLens and BookCrossing datasets for training and testing of the recommenders under evaluation. Results reported for MovieLens-S were obtained from a 5-fold cross-validation on the dataset splits readily available in the zip archive downloaded from the GroupLens website. MovieLens-M was constructed by first selecting all ratings (20 or more) of randomly drawn user ids

Name	Min. User	Max. User	Median User	Avg. User	Min. Item	Max. Item	Median Item	Avg. Item
MovieLens-M	10	1576	6.7×10^1	1.2×10^2	1	2407	9.7×10^1	2.0×10^2
iPlayer	1	56	6.0	7.0	18	147674	2.5×10^3	7.3×10^3
BookCrossing	15	3844	3.0×10^1	6.4×10^1	5	616	9.0	1.5×10^1

Table 3.2: Incidence of test users and items in training set. Reading example: a user in the test set of the of the MovieLens-M dataset rated at least 10, not more than 1576, and on average approx. 120 items; vice versa, all items in the test set of the MovieLens-M dataset have between 1 and 2407 ratings in the training set.

until we reached 1 million ratings. In a second step we assigned randomly 300'000 ratings to the test set under the condition that each item appears at least once in the training set and each test user has at least 10 ratings in the training set. The remaining 700'047 ratings were used for training. The BookCrossing dataset was constructed by first removing ratings for items with a non-alphanumerical id and less than 10 ratings, second we selected all users with twenty or more ratings. From the remaining 369'195 ratings we randomly assigned 110'759 ratings to the test set under the condition that each item in the test set has at least five and each test user at least 15 ratings in the training set. The training set of the iPlayer dataset consists of the viewing logs of the VoD system of one week (February 15-21, 2014) and the test data of the log data of the consecutive week. We considered for both sets, training and test, only viewings with a minimal duration of 5 minutes. The training data comprises 4'703'471 viewings of 808 TV shows and 655'846 users with each show appearing at least 5 times. From the log data of the test week, we randomly selected 5'000 users that were also active during the training week. In the obtained test set with 14'616 viewings each user has at least one viewings during the training time and each item in the test set was watched at least 18 times in the training week. In Table 3.1 we list some characteristics of the used datasets. Figures for the incidence in the training set of users and items used for testing are listed in Table 3.2.

3.1.3 Performance Metrics

Two metrics readily available in the MyMediaLite framework were used to measure accuracy of the recommendation algorithms: area under the ROC curve (AUC) and precision at k (Prec@k). AUC is the number of correctly ordered item pairs consisting of a hit (item that appears in a user's test set) and a no-hit item in the ranked recommendation list divided by the number of all possible pairs of hit and no-hit items. This is the same as the probability that a randomly drawn hit and no-hit item are ordered correctly relative to each other in the recommendation list. Prec@k counts the number of correctly predicted items in the top k items of the recommendation list divided by the cut-off level k. In consideration of the fact that in most applications only a small number of recommendations are presented to the user we decided to calculate Prec@k with small cut-off levels including the top 5, 10 and 20 items.

In order to measure the recommendation quality in the four diversity dimensions

intra-list diversity, coverage, personalization and surprisal introduced in Section 2.3, we extended the MyMediaLite framework with a set of metrics proposed in the literature. All used diversity metrics are top k measures and we applied the same cut-off levels as for Prec@k. For all metrics greater values indicate better diversity in the corresponding dimension. Intra-list diversity and surprisal assess the quality of each recommendation list independent of the other lists. For these metrics we report the scores averaged over all recommendation lists. The personalization score is calculated by comparing each recommendation list to all other generated lists and hence is averaged over all pairs of recommendation lists. Coverage is calculated based on the aggregated top k recommendations for all users.

Intra-list diversity (ILD@k) is measured by the item attributes based diversity metric proposed in Smyth and McClave [2001] that averages the diversity over all item pairs of the top k items $(i_1, ..., i_k)$ in the recommendation list. Top k ILD for a single recommendation list is calculated according to

$$ILD@k = \frac{\sum_{a=1..k} \sum_{b=a..k} (1 - Similarity(i_a, i_b))}{\frac{k}{2} * (k-1)} \tag{3.1}$$

As similarity measure for two items, we used the Jaccard similarity coefficient of each items set of attributes (A_i):

$$Similarity(i_a, i_b) = \frac{|A_a \cap A_b|}{|A_a \cup A_b|} \tag{3.2}$$

The similarity of two items is maximal ($Similarity(i_a, i_b) = 1$) if both items have the same set of attributes and minimal ($Similarity(i_a, i_b) = 0$) if two items do not share any attributes. Since ILD@k is the average of the diversity $(1 - Similarity(i_a, i_b))$ of all item pairs its value is 0 for minimal and 1 for maximal diversity. For the MovieLens datasets the item attributes consist of the classification of the items, here movies, in 19 different genres. A movie can be assigned to more than one genre. The iPlayer dataset has 137 item attributes describing the format of a show (e.g. documentary, game show, or film), its genre (e.g. factual, news, comedy, entertainment, or drama), the subgenre (e.g. politics, biographical, or spoof) and the broadcasting station (e.g. BBC One or BBC Two). Some items have additional attributes for the shows subject, important places or people responsible for or appearing in the show. The BookCrossing dataset does not contain item attribute data and for this reason we do not report ILD figures for this dataset.

To measure coverage we calculated the Gini coefficient (GiniD@k) for the aggregated top k recommendations of all users in the test set (U) as proposed in Adomavicius and Kwon [2012] by

$$GiniD@k = 2 * \sum_{i \in I} \left[\left(\frac{|I| + 1 - rank@k(i)}{|I| + 1} \right) \times \left(\frac{recCount@k(i)}{k * |U|} \right) \right], \tag{3.3}$$

where $|I|$ denotes the cardinality of the set of recommendable items and $recCount@k(i)$ the number of users with item i appearing in the top k recommendations. $rank@k(i)$

is the rank of item i when ordering the items ascending by the number of their appearance among the top k items of all recommendation lists. GiniD@k measures the distribution of the abundance of items in the aggregated recommendation lists. The contribution of a single recommendable item i to the the overall coverage decreases with increasing rank of i and increases if i appears more often in the aggregated lists of recommendations. GiniD@k is equal to 1 if the abundance in the aggregated recommendation lists is the same for each item. In contrast to the original definition of the Gini coefficient used to assess wealth distribution where greater values indicate a more dispersed distribution, GiniD@k increases for a more uniform distribution.

We measured personalization (Pers@k) and surprisal (Surp@k) according to the metrics suggested in Zhou et al. [2010]. Pers@k measures the distinctness of the top k recommendations based on the number of common items averaged over all pairs of generated recommendation sets:

$$Pers@k = \frac{\sum_{a=1..|U|} \sum_{b=a..|U|} (1 - \frac{|topK(u_a) \cap topK(u_b)|}{k})}{\frac{|U|}{2} * (|U| - 1)} \tag{3.4}$$

where U stands for the set of all test users and $topK(u_a)$ denotes the set of the top k items in user a's recommendation list. A value of $Pers@k = 1$ indicates that none of the items appear more than once among the top k items of any two recommendation lists.

Surp@k on the other hand is, like ILD@k, a metric that is calculated separately for each recommendation list and averaged over all users. This metric is based on the rational that recommendations of items of low popularity are perceived by the users as unexpected or surprising. The surprisal value of a single recommendation list is given by

$$Surp@k = \frac{\sum_{a=1..k} \log_2(\frac{|U|}{pop(i_a)})}{k}. \tag{3.5}$$

Here U is the set of training users and $pop(i_a)$ the number of ratings that item a received during the training phase. Assuming the training data lacks duplicate feedbacks, then $\frac{pop(i_a)}{|U|}$ calculates the probability that a randomly selected user rated item a during the training phase and thus item a's self-information [Zhou et al., 2010] or surprisal value S_{i_a} is given by $\log_2(\frac{|U|}{pop(i_a)})$.

Furthermore we measured the performance of all recommenders with two rank sensitive and relevance aware novelty and diversity metrics proposed in Vargas and Castells [2011]: Expected Free Discovery novelty (EFD@k) and Expected Intra-list Distance diversity (EILD@k). Since these metrics also assess the quality of a single recommendation list for a given user we again averaged the results over all users. Even-though the metrics are rank sensitive, which means that a lower ranked item on the recommendation list has a higher impact on the metric score than a higher ranked item, we limited the calculations to the top 5, 10, and 20 positions of the recommendation lists.

EFD@k and EILD@k for a given user u have the general form

$$EFD@k_u, EILD@k_u = C * \sum_{a=1..k} discount(a-1) * relevance(u, i_a) * novDiv(u, i_a, i_{1..k}) \quad (3.6)$$

where i_a is the item on the users recommendation list at position a and C denotes a normalization factor. Rank sensitiveness was considered with the exponential discount model

$$discount(a) = c^a \quad (3.7)$$

with $c = 0.85$, which is the same value as used in Vargas and Castells [2011]. A relevance aware diversity or novelty metric considers also the recommender's prediction power in the metric score. In the implicit feedback case this can be implemented with a simple boolean function

$$relevance(u, i) = \begin{cases} 1, & i \in T_u \\ 0, & i \notin T_u \end{cases} \quad (3.8)$$

where T_u is the set of all test items of user u. EFD and EILD share the definitions for $discount(r)$ and $relevance(u, i)$ and distinguish only in the novelty model ($novDiv(u, i_a, i_{1..k})$). In EFD the novelty score is obtained from the items self-information value S_{i_a}, which makes EFD a rank sensitive and relevance aware version of the Surp@k measure. While EFD uses a popularity-based novelty model EILD is based on a item distance diversity model calculated from the Jaccard coefficient of the recommendation lists top k items attribute sets, see Equation (3.2). Hence EILD can be considered as a rank sensitive and relevance aware version of the ILD@k measure.

3.2 Accuracy and Diversity of Graph-based Recommenders

We used the metrics defined in Section 3.1 to determine the performance of the graph-based recommendation algorithms introduced in Section 1.2 and the baseline recommenders (Section 3.1). The results of this first round of evaluation are summarized in Table 3.3. Our experiments corroborate the findings presented in Cooper et al. [2014]: Ranking the items by the entries of the third power of the vertex transition probability matrix (P^3 and P^3_α) is the most accurate graph-based recommendation algorithm. This finding is consistent through all four datasets and both types of accuracy metrics. We also found only a small difference in the performance figures obtained with simulation of random walks (S) compared to results from matrix algebra calculations (M), i.e., P^3 (S) is similar to P^3 (M), P^5 (S) is similar to P^5 (M), and IR (S) is similar to IR (M). In accordance with the results of Cooper et al. [2014], we found recommendation lists of comparable AUC accuracy for L^+ when compared to P^3. The precision of L^+, however, is lower at all three measured thresholds for all datasets. As reported by Cooper et al. [2014], we found in our experiments on the MovieLens-M

P^3, P^3_α, and L^+ most accurate graph-based recommenders

	Recommender	AUC*	Prec*	ILD	GiniD	Pers	Surp	EILD	EFD
MovieLens-S	Perfect	100.0	100.0	0.817	0.233	0.824	3.88	0.736	7.92
	Random	49.7	3.4	0.828	0.824	0.985	5.19	0.141	0.23
	MostPop	82.7	27.0	0.815	0.024	0.434	1.61	0.555	1.73
	I-kNN (k=100)	90.6	36.2	0.789	0.244	0.943	3.00	0.542	2.52
	WI-kNN (k=100)	91.1	44.2	0.806	0.127	0.890	2.34	0.627	3.12
	BPRMF (d=10)	91.5	41.8	0.794	0.196	0.916	2.65	0.604	2.94
	3Path	85.6	34.4	0.820	0.033	0.578	1.71	0.606	2.40
	IB-3Path	85.6	34.4	0.820	0.033	0.578	1.71	0.606	2.40
	P^3 (S)	88.7	38.4	0.821	0.044	0.684	1.81	0.636	2.57
	P^3 (M)	88.8	38.6	0.820	0.043	0.678	1.80	0.637	2.57
	P^5 (S)	85.5	32.0	0.821	0.031	0.551	1.67	0.602	2.09
	P^5 (M)	85.7	32.3	0.820	0.030	0.540	1.66	0.604	2.10
	P^3_α (M, $\alpha = 1.7$)	89.6	36.4	0.819	0.051	0.699	1.96	0.627	2.38
	IR (S)	81.9	28.8	0.819	0.028	0.504	1.70	0.519	1.89
	IR (M)	82.0	29.1	0.818	0.026	0.458	1.69	0.524	1.92
	BnB	79.4	27.4	0.815	0.113	0.855	2.23	0.526	1.89
	OW (S)	82.7	27.1	0.814	0.025	0.442	1.61	0.556	1.74
	CT (S)	82.7	27.2	0.815	0.026	0.464	1.62	0.555	1.74
	L^+ (M)	88.8	37.0	0.788	0.198	0.929	3.06	0.527	2.63
MovieLens-M	Perfect	100.0	100.0	0.795	0.218	0.927	4.01	0.712	9.90
	Random	50.2	1.8	0.832	0.900	0.994	6.33	0.101	0.16
	MostPop	85.1	25.1	0.810	0.009	0.401	1.84	0.523	2.09
	I-kNN (k=150)	91.4	33.9	0.669	0.221	0.973	3.62	0.476	3.20
	WI-kNN (k=150)	91.8	42.3	0.699	0.090	0.918	2.75	0.546	3.93
	BPRMF (d=50)	92.1	38.8	0.726	0.189	0.952	3.22	0.567	3.65
	3Path	86.7	28.0	0.803	0.010	0.449	1.86	0.559	2.48
	IB-3Path	86.7	28.0	0.803	0.010	0.449	1.86	0.559	2.48
	P^3 (S)	88.9	30.2	0.809	0.011	0.511	1.89	0.602	2.71
	P^3 (M)	89.1	30.2	0.809	0.011	0.497	1.88	0.604	2.73
	P^5 (S)	85.8	25.7	0.811	0.010	0.432	1.84	0.537	2.17
	P^5 (M)	86.0	26.0	0.812	0.009	0.410	1.84	0.537	2.19
	P^3_α (M, $\alpha = 1.8$)	90.3	31.0	0.811	0.027	0.644	2.13	0.614	2.81
	IR (M)	85.1	25.8	0.802	0.009	0.405	1.85	0.521	2.18
	BnB	78.6	15.2	0.807	0.088	0.925	2.78	0.431	1.38
	L^+ (M)	88.1	25.7	0.577	0.218	0.971	4.22	0.378	2.67
iPlayer	Perfect	100.0	100.0	0.857	0.068	0.172	8.21	0.696	2.61
	Random	51.5	3.1	0.812	0.954	0.968	8.01	0.031	0.03
	MostPop	78.0	19.8	0.861	0.038	0.163	3.31	0.193	0.17
	I-kNN (k=50)	89.0	27.9	0.704	0.340	0.867	6.11	0.159	0.33
	WI-kNN (k=150)	92.6	48.1	0.724	0.195	0.734	4.55	0.326	0.70
	BPRMF (d=50)	91.0	46.8	0.771	0.211	0.734	4.54	0.324	0.62
	3Path	89.7	36.2	0.808	0.077	0.490	3.75	0.302	0.41
	IB-3Path	89.7	36.2	0.808	0.077	0.490	3.75	0.302	0.41
	P^3 (S)	91.2	41.2	0.803	0.109	0.569	3.96	0.321	0.49
	P^5 (S)	89.2	33.2	0.836	0.060	0.420	3.60	0.282	0.32
	P^3_α (T, $\alpha = 1.5$)	91.5	42.6	0.792	0.139	0.617	4.13	0.319	0.53
	IR (S)	84.9	29.1	0.855	0.042	0.266	3.37	0.255	0.24
	IR (M)	85.1	29.1	0.855	0.042	0.259	3.36	0.256	0.24
BookCrossing	Perfect	100.0	100.0	N/A	0.266	0.828	7.80	N/A	11.43
	Random	50.1	0.2	N/A	0.748	0.999	8.57	N/A	0.02
	MostPop	71.8	5.1	N/A	0.001	0.111	3.95	N/A	0.41
	I-kNN (k=800)	75.3	7.2	N/A	0.148	0.976	7.38	N/A	0.67
	WI-kNN (k=1600)	78.1	10.0	N/A	0.150	0.955	6.79	N/A	1.06
	BPRMF (d=10)	79.8	5.2	N/A	0.100	0.966	6.39	N/A	0.45
	3Path	77.8	7.2	N/A	0.002	0.436	4.14	N/A	0.62
	IB-3Path	77.8	7.2	N/A	0.002	0.434	4.14	N/A	0.62
	P^3 (S)	81.7	9.0	N/A	0.016	0.668	4.58	N/A	0.77
	P^3 (M)	82.5	9.1	N/A	0.015	0.652	4.56	N/A	0.78
	P^5 (S)	78.8	6.2	N/A	0.002	0.355	4.12	N/A	0.51
	P^5 (M)	80.6	6.3	N/A	0.002	0.271	4.09	N/A	0.52
	P^3_α (M, $\alpha = 0.9$)	82.5	9.0	N/A	0.010	0.610	4.44	N/A	0.77
	P^5_α (M, $\alpha = 1.3$)	81.4	6.9	N/A	0.006	0.426	4.34	N/A	0.57
	BnB	66.7	3.5	N/A	0.041	0.923	5.36	N/A	0.32
	L^+ (M)	82.3	5.0	N/A	0.318	0.996	9.19	N/A	0.60

Table 3.3: Accuracy and diversity performance of graph-based (yellow background), personalized (green), and non-personalized (blue) baseline recommendation algorithms. Due to space restrictions we include only the measurements for Prec, ILD, GiniD, Pers, Surp, EILD, and EFD at a cut-off value of 20 ($k = 20$). Furthermore, we list only the figures for the parameter setting that achieved the highest AUC performance. The parameter k of the nearest neighbors-based methods (I-kNN and WI-kNN) denotes the neighborhood size and the parameter d of the matrix factorization-based technique (BPRMF) denotes the number of latent factors in the model. ***Accuracy performance figures (AUC and Prec) are given in percentage of the Perfect recommender.*

and iPlayer datasets that the parametrized version of P^3, P^3_α provides improved accuracy (both AUC and precision) over the pure transition probability-based P^3 method. For MovieLens-S P^3_α with a value of α that optimizes AUC performance shows a lower Prec@k performance than P^3 at all three measured threshold levels. With the same α value the AUC performance of P^3_α is better than of P^3. We could not observe an improved accuracy for P^3_α when compared to P^3 in the experiments on the BookCrossing dataset. If we compare the performance of graph-based recommenders to the baseline methods on the MovieLens and iPlayer datasets, we find that even the best performing graph-based recommenders give less accurate recommendations than at least one of the baseline recommenders. For the BookCrossing dataset P^3 and P^3_α are the methods with the overall highest AUC values. But if we compare accuracy by Prec@20 the nearest neighbor-based baseline methods again outperform P^3 and P^3_α, i.e., maximal Prec@20 for WI-kNN is 12.5 (neighborhood size of 50) and maximal precision of P^3_α is 9.1 (with $\alpha = 1.1$, results not included in Table 3.3). Our results contradict earlier reported results [Cooper et al., 2014; Gori and Pucci, 2007] regarding the accuracy of the ItemRank (IR) recommender: in our experiments with the MovieLens-S dataset, the accuracy of this method is lower than for OW or CT and also clearly worse than for vertex transition probability-based methods or L^+.

Worse accuracy of graph-based recommenders compared to baselines

Now let us compare the performance of graph-based recommenders for the four diversity dimensions (see columns 5-8 of Table 3.3): the best performing graph-based recommender in terms of coverage (GiniD), personalization (Pers) and surprisal (Surp) is L^+, followed by the very inaccurate BnB method and, with a huge margin, by P^3_α and P^3. The intra-list diversity (ILD) of recommendation lists from L^+, on the other side, is worse than for any other method but good for P^3 and P^3_α. Except for L^+, the most accurate graph-based recommenders are much worse than the personalized baseline methods in terms of coverage, personalization, and surprisal. P^3_α and P^3 generate similar recommendation lists for different users (low Pers) with a small number of different items (low GiniD) of high popularity (low Surp) at the top. This performance profile indicates that the item ranking of P^3 and P^3_α is similar to the one produced by MostPop (MostPop has a Pers> 0 because items already collected by the user are filtered out from the recommendation list). We also interpret the rather good ILD performance of many graph-based recommenders as indication for the strong influence of item popularity on the ranking list since the ranking according to item popularities (MostPop) results also in a recommendation list with good intra-list diversity. Introducing rank sensitiveness and relevance awareness in the ILD (EILD) and Surp (EFD) metrics does not qualitatively change the findings of our performance assessment (see last two columns of Table 3.3): the recommendation novelty of the best graph-based recommenders measured by EFD is worse than for the best personalized baseline algorithms in all datasets while recommendation diversity in terms of EILD is comparable.

Worse coverage, personalization, and surprisal of most graph-based recommenders compared to baselines

Item ranking of most graph-based recommenders is strongly influenced by item popularity

3.3 Performance Impact of Popularity Normalization

Using our insights from the performance evaluation of graph-based recommendation algorithms proposed in the literature (see Section 3.2), we developed a re-ranking procedure to reduce the impact of item popularity on the item ranking (see Section 2.2). In this section we present the performance improvements achieved with popularity normalization for graph-based recommendation algorithms based on vertex transition probabilities. We measured the accuracy and diversity performance of AN-P^3, RN-P^3, AN-P^5, RN-P^5, AN-IB-3Path and RN-IB-3Path for values of the normalization strength determining factor β in the range from -0.2 to 1.2 in steps of 0.1. Hence we also measured the performance of the non-normalized ($\beta = 0$) methods P^3, P^5, and IB-3Path again. In all experiments the AN-methods showed better performance in terms of accuracy and all four diversity dimensions than the RN-methods. For this reason we will exclude the RN-methods from the following discussion.

3.3.1 Popularity Normalization Improves Accuracy

In contrast to our expectations, the experiments revealed a clear accuracy improvement for all three AN-methods with small positive values of β when compared to the non-normalized method (see Figure 3.1). The only exception to this finding is the performance of AN-P^3 and AN-IB-3Path on the BookCrossing dataset for which the AUC remained unchanged for small positive values of β (see Figure 3.1 A4). For all datasets the highest precision (Prec@20) was observed for popularity normalized transition probabilities of random walks of length three (AN-P^3, see Figure 3.1 B1-B4). The same method shows also good AUC accuracy and is only outperformed on the BookCrossing data by AN-P^5 (see Figure 3.1 A1-A4). For the MovieLens and iPlayer dataset both accuracy scores, AUC and Prec@20, steadily increase for AN-P^3 with increasing values of β and reach maximums between 0.6 and 0.8. Increasing β further ($\beta > 0.8$) leads to a very pronounced drop in accuracy. We think it is worth noting that the computationally cheaper AN-IB-3Path method achieved for the iPlayer dataset almost the same level of accuracy as AN-P^3.

Accuracy of AN-P^3 is comparable to the performance of the baseline methods (see Section 3.5)

Accuracy of popularity normalized methods first increases and then drops dramatically for increasing values of β

AN-P^3 is also the best performing algorithm in terms of precision for the BookCrossing dataset (Figure 3.1 B4). Compared to the other datasets we found the Prec@20 maximum for AN-P^3 at a considerably smaller β value of 0.3. The smaller accuracy maximizing β value for AN-P^3 can be explained by the generally smaller item popularity difference for this dataset and the great share of items in a narrow popularity range (approx. 80% of the items received between 5 and 15 ratings, see also Table 3.1). This dataset structure reduces the probability that a random walk ends at high degree vertex and hence defeats the purpose of popularity normalization.

For the BookCrossing dataset, AN-P^3 shows the highest Prec@20 score and AN-P^5 the best AUC performance (Figure 3.1 A4 and B4). This means that AN-P^3 is better in placing relevant items at the top of the recommendation list while AN-P^5 is better in distinguishing between relevant and irrelevant items, i.e., with AN-P^5 a user needs to retrieve less items to obtain all relevant items than with AN-P^3.

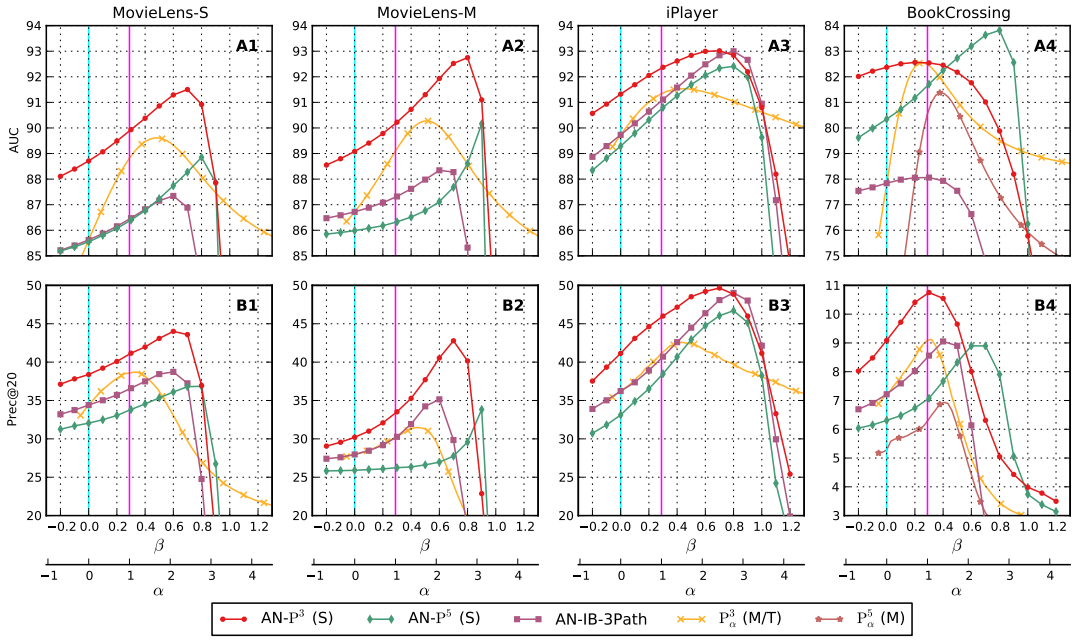


Figure 3.1: Impact of transition probability tuning (P^3_α and P^5_α) and popularity normalization (AN-P^3, AN-P^5, and AN-IB-3Path) on prediction accuracy in dependence of α and β, respectively. Note that P^3_α with $\alpha = 1$, P^3, and AN-P^3 with $\beta = 0$ describe all the same algorithm, correspondingly P^5_α with $\alpha = 1$, P^5, and AN-P^5 with $\beta = 0$ are also equivalent. Furthermore AN-IB-3Path with $\beta = 0$ is the same method as P^3_α with $\alpha = 0$. In order to facilitate comparison of identical algorithms, each subplot contains a vertical line colored in cyan at $\beta = 0$ and a vertical line colored in magenta at $\alpha = 1$.

We can explain this performance difference by considering the structure of the graph constructed from the BookCrossing training set and the implications of longer random walks. The BookCrossing graph has a greater diameter compared to the graphs of the other datasets (see Table 3.1) hence longer walks are required to obtain the ranking weights for all items. Some of the items that are discovered in walks of length five but not in walks of length three are relevant even-though they are faraway from the target user. With AN-P^3 these items are placed at the very bottom of the recommendation list due to the lack of ranking weights, but can populate higher ranks with AN-P^5, which results in the better AUC score for AN-P^5. However, the additionally discovered relevant items are likely to be placed at low positions of the recommendation list and are therefore of little value from a user's perspective. The precision, on the other side, of a transition probability-based recommender is likely to drop with increasing walk lengths because the recommendation list is constructed by considering user profiles of non-immediate neighbors that are less similar to the target user than the users in two edge distance.

Better precision with shorter random walks

3.3.2 Popularity Normalization Improves Coverage, Personalization, and Surprisal

Increasing item space coverage (GiniD) and reducing the dominance of high popularity items at the top of the recommendation list (Surp) are the primary goals of popularity normalization. According to the plots presented in Figure 3.2, the absolute popularity normalization (AN) procedure fulfills these intents: For all AN-methods and datasets we could find values of β with better GiniD (see Figure 3.2 B1-B4) and Surp (Figure 3.2 D1-D4) than the non-normalized methods. In terms of coverage, AN-P^3 is again the best performing method with better GiniD scores for any value of β than all other AN-methods. What we find interesting is the pronounced drop in coverage for high values of β indicating that the top places of the recommendation lists become dominated by low popularity items. With regard to the surprisal metric (Figure 3.2 D1-D4) none of the popularity normalized method clearly outperforms the others. But AN-P^3 seems to be favorable since it shows the smoothest transition (smaller curvature) from high popularity ($\beta \leq 0$) to low popularity items ($\beta > 1.0$) dominated recommendation lists. AN-P^3 achieved also the highest personalization scores in all experiments (Figure 3.2 C1-C4). The changes of Pers in dependency of β is for all methods similar to the impact of β on coverage, consequentially we detected the maximum Pers and GiniD scores for the same parameter values. In contrast to the other datasets, we observed for the BookCrossing dataset only a marginal drop in personalization for AN-P^3 and AN-P^5 for values of β greater than the value of β that gives maximal coverage (see Figure 3.2 B4 & C4). For the same dataset, the recommendation lists of AN-P^3 with $\beta > 0.6$ and of AN-P^5 with $\beta > 0.9$ are almost perfectly personalized (Figure 3.2 C4). This observations can be explained by the much greater number of items than users in the BookCrossing dataset.

Only low popularity items are recommended for $\beta > 1$

Similar *relative* coverage, personalization, and surprisal performance for different recommenders

The ILD metric is the only diversity performance measure that shows a lower score for positive values of β (Figure 3.2 A1-A3). This means that a single recommendation list contains more items that have the same attribute values assigned, e.g., more movies of the same genre. Together with the finding of better accuracy and increased distinctness of recommendation lists (improved Pers), we can follow that recommendations from AN-methods with small positive values of β better match a user's preference profile according to the training items. We leave it as an open question to analyze to what extent the drop in ILD is correlated with the user's perceived diversity of a recommendation list. The ILD scores increase again for high degrees of popularity normalization, e.g., for $\beta > 1$ in the experiments with the iPlayer dataset (Figure 3.2 A3). Because a great share of all items are low popularity items, we can assume that a great variety of item attribute values is present among all items with low popularity. Under this assumption the increase in ILD for strong popularity normalization can be explained by recommendation lists with a nearly random (low accuracy) selection of low popularity items placed at the top.

Better accuracy reduces intra-list diversity

In contrast to the ILD performance, popularity normalization with small positive values of β increases scores of EILD, the rank sensitive and relevance aware version of ILD (see Figure 3.3 A1-A3). The improved EILD performance corresponds to the

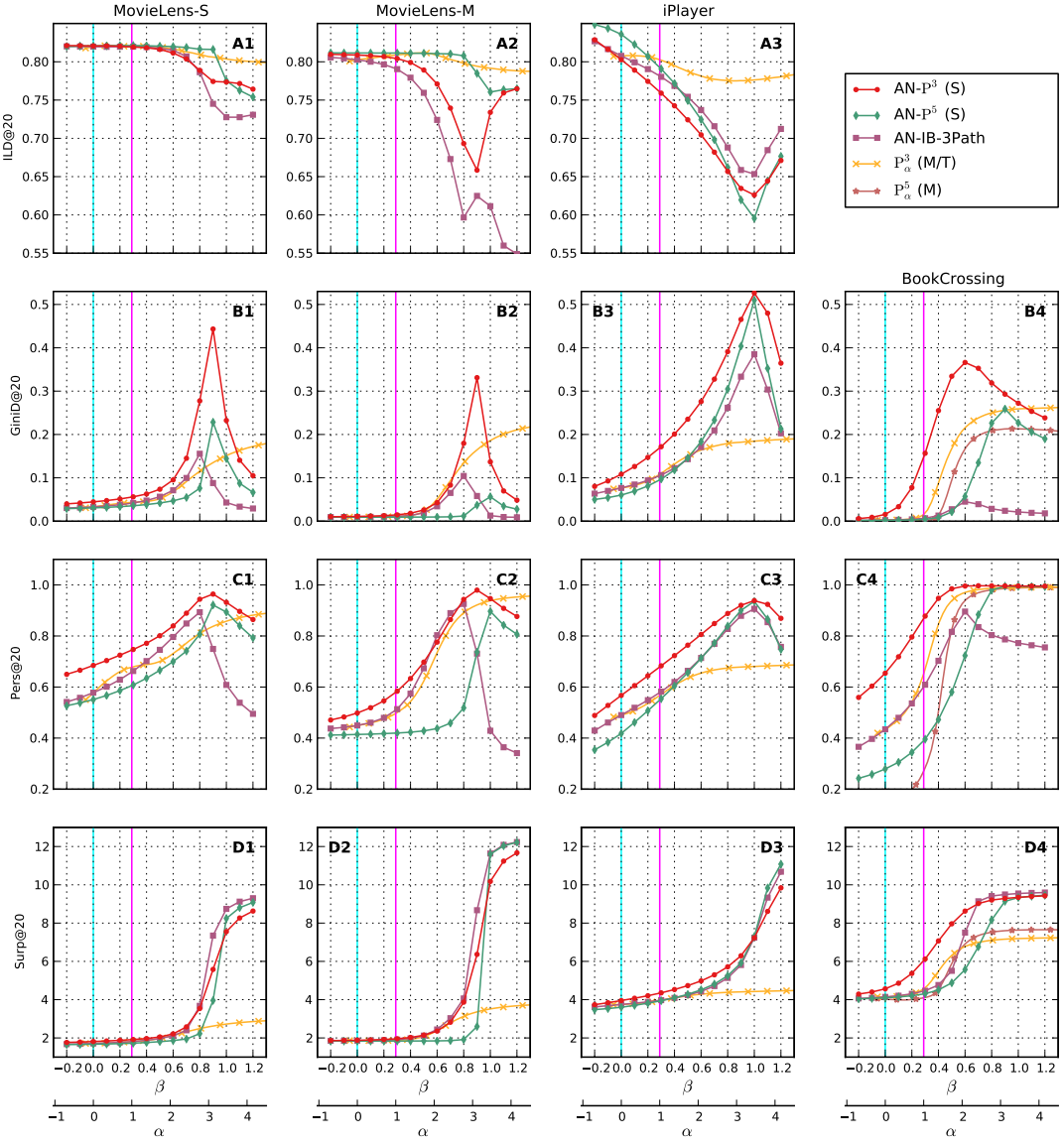


Figure 3.2: Impact of transition probability tuning and popularity normalization on intra-list diversity, coverage, personalization, and surprisal in dependence of α and β, respectively.

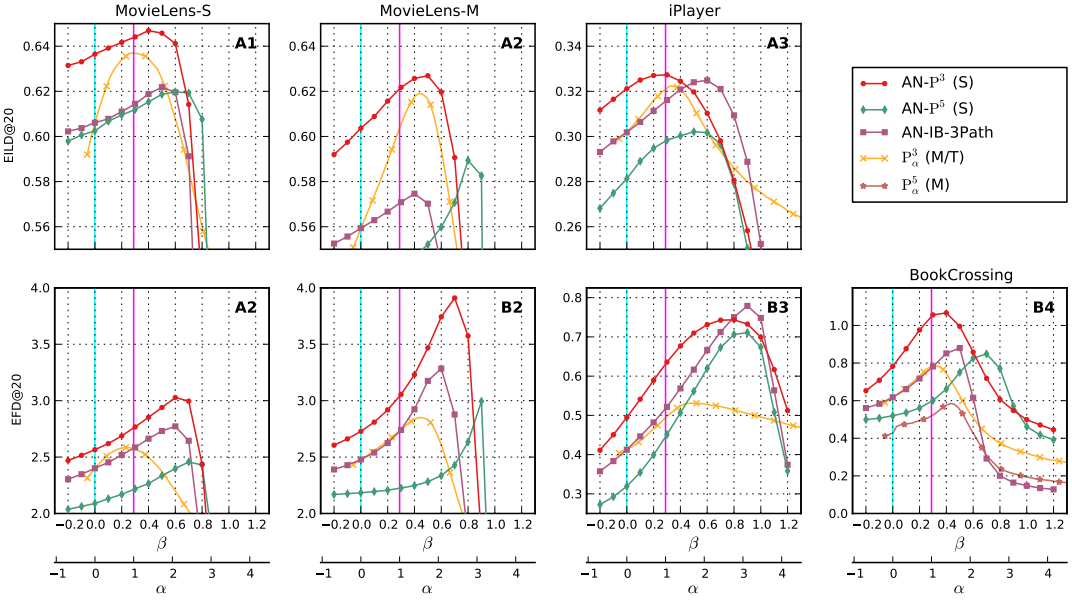


Figure 3.3: Impact of transition probability tuning and popularity normalization on rank sensitive and relevance aware novelty and diversity in dependence of α and β, respectively.

better precision of AN-methods. We also found a steep drop in EILD performance for greater values of β as it is the case for precision. The drop in EILD occurs at smaller values of β (e.g. at $\beta = 0.6$ for AN-P^3 on the MovieLens-M dataset, Figure 3.3 A2) than the drop in precision ($\beta = 0.8$, Figure 3.1 B2), which can be explained by the increasing negative impact of the reduced intra-list diversity for greater β's. The underlying metrics of EFD, Surp and precision, both increase for small values in β hence we could observe a more pronounced increase for EFD than for EILD (see Figure 3.3 B1-B4). Again the EFD performance drops dramatically with a loss in precision for $\beta > 0.8$. For both rank sensitive and relevance aware metrics, AN-P^3 outperforms the other two AN-methods on the MovieLens and BookCrossing datasets. On the iPlyer dataset AN-IB-3Path shows a slightly better EFD performance than AN-P^3.

3.4 Popularity Normalization versus Transition Probability Tuning

In this section we compare the performance of the popularity normalized methods AN-P^3 and AN-P^5 against the improvements achievable by tuning transition probabilities (P$^3_\alpha$ and P$^5_\alpha$) as proposed in Cooper et al. [2014]. As already described in Subsection 3.3.1, the accuracy of AN-methods increases for small positive values of β followed by a pronounced drop. We observed the same behavior for P$^3_\alpha$ when we varied parameter α over the range of -0.2 to 4.5 in steps of 0.1. The increase in accuracy for P$^3_\alpha$ over the

AN-P^3 is more precise than P$^3_\alpha$

non-parametrized method P^3 (equivalent to P^3_α with $\alpha = 1$) is small (MovieLens and iPlayer, see Figure 3.1 A1-A3 & B1-B3) to none observable (BookCrossing, Figure 3.1 A4 & B4). In the experiments with the MovieLens and iPlayer datasets, we obtained more accurate (better AUC and Prec@20) recommendation lists for AN-P^3 than P^3_α (Figure 3.1 A1-A3 & B1-B3 and Table 3.4). For the BookCrossing data the AUC performance (Figure 3.1 A4) of P^3_α and AN-P^3 are comparable but the precision (Figure 3.1 B4) of AN-P^3 is considerably higher.

We assume that the the random walk length of vertex transition probability-based recommenders needs to be increased with an increasing diameter of the queried user-item feedback graph. Since the BookCrossing network has a greater diameter (see Table 3.1) than the graphs constructed from the training data of the other datasets we measured for BookCrossing also the performance of P^5_α for the same values of α as for P^3_α. While AN-P^5 outperforms AN-P^3 in the AUC metric we could not observe a similar relative performance shift between P^5_α and P^3_α (Figure 3.1 A4).

Increasing the parameter α of P^3_α and P^5_α results in an increased attenuation of a user's preference propagation. This means that the ranking of an item increases if the paths from a user to the item consists of low degree vertices. For this reason we would expect to see less popular items appearing at higher positions in the recommendation lists, which in turn means improved coverage and surprisal scores for greater values of alpha. Our experiments support this hypothesis (see Figure 3.2). For P^3_α we found greater values for GiniD (see Figure 3.2 B1-B4) and Surp (Figure 3.2 D1-D4) with increasing values of α for all datasets. However, the improvements achieved with AN-P^3 for these metrics were higher. The increase in personalization (see Figure 3.2 C1-B4) of P^3_α is comparable to AN-P^3, except for the iPlayer dataset, where we observed a better personalization for all AN-methods (Figure 3.2 C3). As for the AN-methods, ILD performance (Figure 3.2 A1-A3) of P^3_α drops with increasing values of α but the loss in ILD is smaller for P^3_α than for any AN-methods. We can explain the superior ILD performance of P^3_α for values of $\alpha > 1$ by the low Surp gain of P^3_α. This indicates that the recommendation lists of P^3_α are still dominated by high popularity items, which also leads to good intra-list diversity as it is shown by the performance of the MostPop recommender. The EILD and EFD performances mirror the image of the accuracy changes upon varying α and β. For both metrics and all four datasets we found values of β where AN-P^3 gives better results than the best performance observed for P^3_α.

AN-P^3 gives better coverage, personalization, and surprisal than P^3_α

The accuracy and diversity performance of P^3_α and AN-P^3 with α and β parameter values that maximize AUC performance are listed in Table 3.4. At the maximal level of AUC performance AN-P^3 outperforms P^3_α in terms of AUC, Prec@20, GiniD@20, Pers@20, Surp@20, and EFD@20 for all datasets. P^3_α shows better ILD@20 and EILD@20 performance. All performance differences are statistically significant (Wilcoxon $p<0.0001$) except for the difference in EILD on the MovieLens-S data (Wilcoxon $p=0.053$, see Appendix A.3).

Performance differences between P^3_α and AN-P^3 are significant

	MovieLens-S		MovieLens-M		iPlayer		BookCrossing	
	P^3_α (M)	AN-P^3 (T)	P^3_α (M)	AN-P^3 (T)	P^3_α (T)	AN-P^3 (T)	P^3_α (T)	AN-P^3 (T)
α	1.7	1.0	1.8	1.0	1.5	1.0	0.9	1.0
β	0.0	0.7	0.0	0.8	0.0	0.7	0.0	0.2
AUC*	89.6	91.9	90.3	92.9	88.0	89.5	82.5	82.7
Prec@20*	36.4	45.1	31.0	40.8	42.6	49.6	9.0	10.4
ILD@20	81.9	80.0	0.811	0.691	0.792	0.682	N/A	N/A
GiniD@20	0.051	0.128	0.027	0.172	0.139	0.327	0.010	0.076
Pers@20	0.699	0.879	0.644	0.941	0.617	0.848	0.610	0.795
Surp@20	1.96	2.49	2.13	3.79	4.13	5.29	4.44	5.36
EILD@20	0.627	0.620	0.614	0.516	0.319	0.298	N/A	N/A
EFD@20	2.38	3.11	2.81	3.64	0.530	0.743	0.770	0.977

Table 3.4: Accuracy and diversity of P^3_α and AN-P^3 at level of maximal AUC performance. See Appendix A.3 for a significance test of the performance differences of the per-user metrics (AUC, Prec, ILD, Surp, EILD, and EFD). **Accuracy performance figures (AUC and Prec@20) are given in percentage of the Perfect recommender.*

3.5 Trading-off between Accuracy and Diversity with Popularity Normalization

Sacrificing on accuracy for increased recommendation diversity can lead to improved user satisfaction [Ziegler et al., 2005]. Therefore we think that a recommender should allow to find a good trade-off between accuracy and diversity. As described in Section 3.3, tuning the parameter β of popularity normalized algorithms allows to maximize either recommendation accuracy or diversity. In Figure 3.4 and 3.5 we show the achievable diversity performance at a given level of accuracy of vertex transition probabilities-based recommenders. In each of the subplots of Figure 3.4 and 3.5 we also indicate the diversity performance at the highest achieved accuracy level (AUC in Figure 3.4 and Prec@20 in Figure 3.5) for each of the personalized baseline algorithms (I-kNN, WI-kNN, and BPRMF). An ideal recommender would occupy the upper right corner of each subplot, generating recommendations that are both, accurate and diverse. The results for MovieLens-S are qualitatively equivalent to the results obtained with MovieLens-M if not stated otherwise. For this reason and for the sake of clarity we omitted the plots for MovieLens-S in the Figures 3.4 and 3.5.

With the appropriate value of β AN-P^3 gives for most datasets and metric combinations the best trade-off between accuracy and diversity of all vertex transition probability-based recommenders. The data points in Figure 3.4 and 3.5 for AN-P^3 represent the accuracy and diversity performance at different values of β. In all subplots describe the data points of AN-P^3 a curve between low accuracy and diversity and low accuracy and high diversity, passing a point of infinite slope with the highest accuracy and moderate diversity. Thanks to the transition from low diversity to high diversity through the accuracy maximum, popularity normalization seems to be an appropriate procedure to find an optimal trade-off between accuracy and diversity. For the coverage, personalization, and surprisal diversity dimensions, increasing values of β result in higher diversity scores. Intra-list diversity, on the other hand, increases for smaller values of β (neglecting values of β outside the range that gives reasonable accuracy). Due to this diverging influence of increasing β values on the four diver-

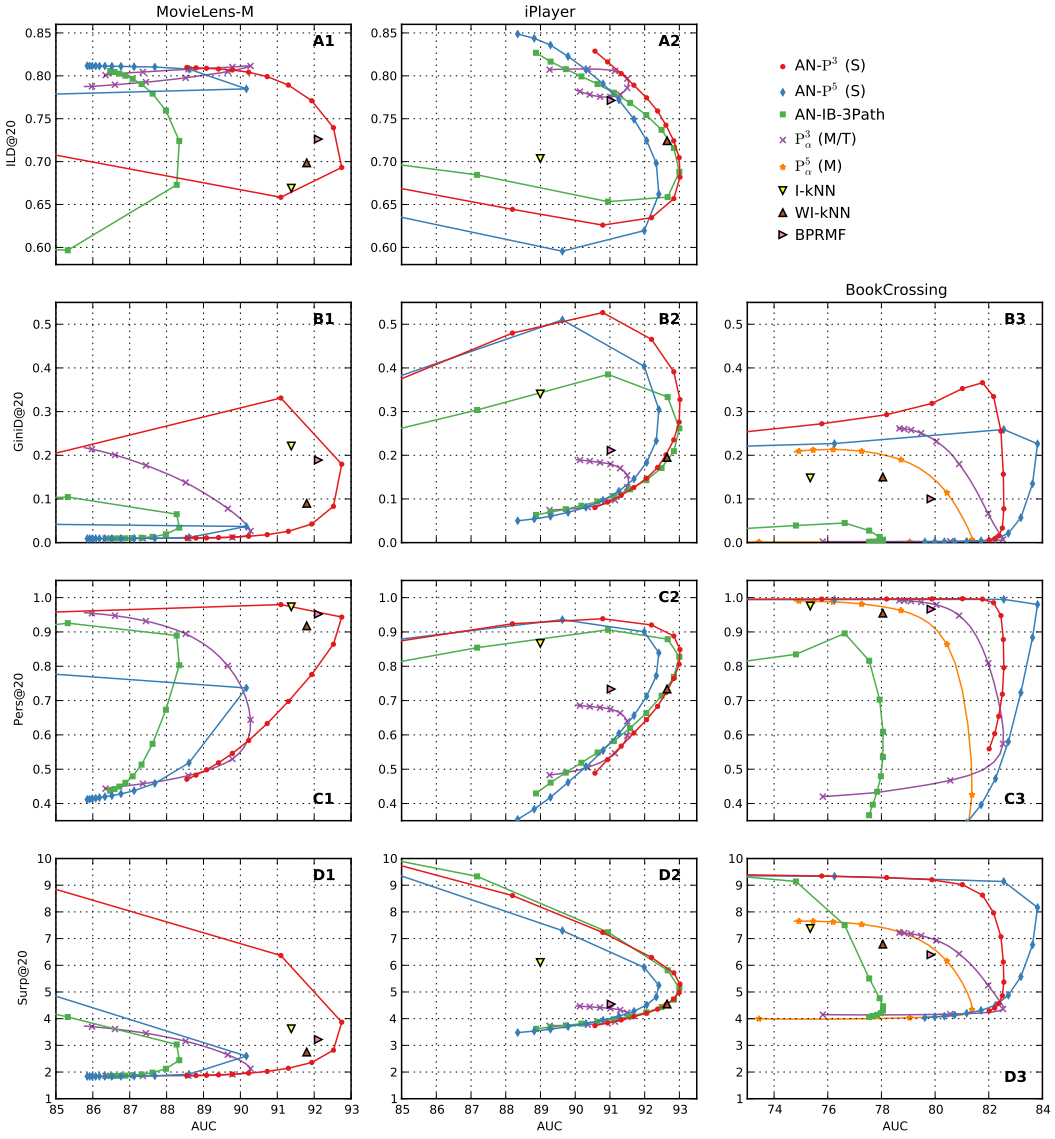


Figure 3.4: Trade-off between AUC and the four diversity dimensions. The curves for P^3_α and P^5_α show the performance of the recommenders at varying values of α. The curves for AN-P^3, AN-P^5, and AN-IB-3Path show the performance of the recommenders at varying values of β. For the baseline algorithms we show the results for the parameter settings (neighborhood size for I-kNN and WI-kNN and number of latent factors for BPRMF) that gave the best AUC performance.

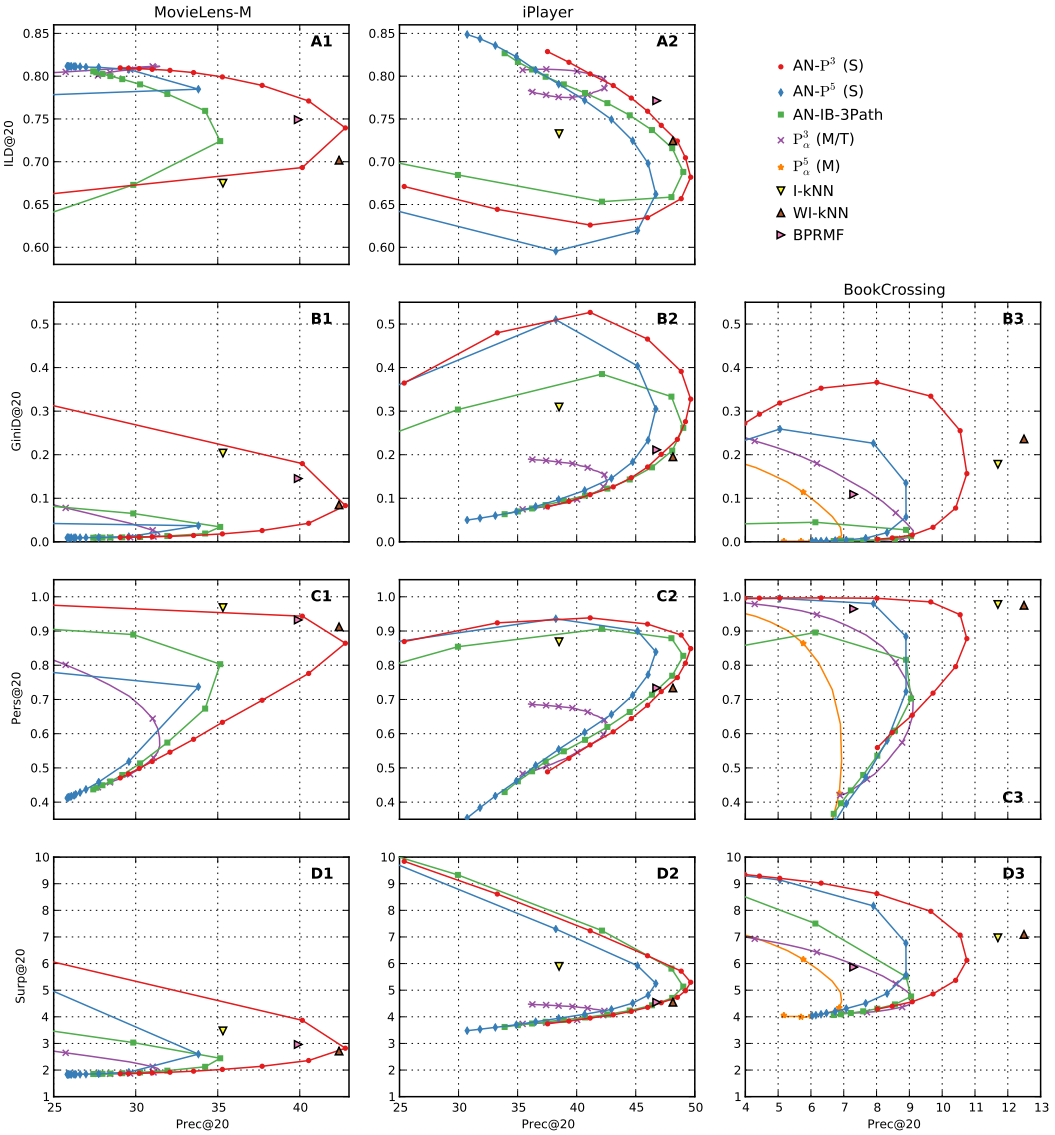


Figure 3.5: Trade-off between Prec@20 and the four diversity dimensions. For the baseline algorithms we show the results for the parameter settings that gave the best Prec@20 performance.

sity dimension it is not only necessary to find an optimal trade-off between diversity and accuracy but also between intra-list diversity and coverage, personalization, and surprisal

When we compare the maximal achieved accuracy of AN-P^3 on the MovieLens and iPlayer data with the accuracy of the most accurate baseline recommenders we find similar or better scores for AN-P^3 in both metrics (AUC and Prec@20). On the BookCrossing dataset, the AUC performance (see Figure 3.4 B3-D3) of AN-P^3 is better than for all baseline algorithms but both nearest neighbor algorithms achieved with small neighborhood sizes ($k = 10$ for I-kNN and $k = 50$ for WI-kNN) better precision scores (see Figure 3.5 B3-D3) than AN-P^3. The best precision on the BookCrossing dataset was found for WI-kNN (Prec@20 = 12.49 with $k = 50$). Increasing the neighborhood size of WI-kNN results in a steady decrease in precision (Prec@20 = 9.98 with $k = 1600$). Note that this drop in precision was only observed for the BookCrossing dataset and that the AUC performance of WI-kNN on the same data follows the opposite trend. AN-P^3 is similar to a kNN procedure with a large k since with random walks of length three many different user profiles are considered for item ranking, i.e., the profiles of all users that rated at least one item that also the target user rated. This explains the lower maximal precision of AN-P^3 compared to a kNN recommender with small k that uses a smaller number of users that are more similar to the target user for recommendation generation.

AN-P^3 is at least as accurate as the baseline methods for datasets with fewer items than users

Next we compare the recommendation diversity of AN-P^3 to the diversity obtained with the most accurate baseline recommender. AN-P^3 did not achieve a comparable precision for any value of β as the best baseline method (WI-kNN) on the BookCrossing dataset (see Figure 3.5 B3-D3). For this reason we can not compare their difference in diversity and hence we will not consider the precision versus diversity performance of AN-P^3 on the BookCrossing data in the following discussion. Coverage and surprisal of the baseline recommender with the best AUC performance is comparable (MovieLens-S) or lower (MovieLens-M, iPlayer, and BookCrossing, see Figure 3.4 B1-B3) than the coverage of AN-P^3 at the same level of AUC performance. The score for personalization of the baseline method with the best AUC value is in the same range for MovieLens-S, MovieLens-M, and BookCrossing (Figure 3.4 C1 & C3) and lower for iPlayer (Figure 3.4 C2) as the personalization score of AN-P^3 at the same AUC accuracy. When comparing intra-list diversity at the level of the best baseline AUC score we find that AN-P^3 gives comparable performance for the MovieLens-S and iPlayer (Figure 3.4 A2) and a slightly better performance for the MovieLens-M (Figure 3.4 A1) dataset. The performance of AN-P^3 is generally less advantageous when we compare Prec@20, instead of AUC, versus diversity. In particular, we could not observe a better coverage and surprisal for AN-P^3 than for the baseline method in the MovieLens-M experiments with the best Prec@20 score (WI-kNN, Figure 3.5 B1 & D1). For the same dataset gives AN-P^3 at the precision level of WI-kNN also worse personalization (Figure 3.5 C1). For the iPlayer dataset is the precision of AN-P^3 lower at the level of intra-list diversity of the baseline recommender with the best ILD@20 score (BPRMF, Figure 3.5 A2).

AN-P^3 gives better or comparable diversity than baselines at same level of AUC accuracy

Considering the results obtained with the iPlayer and MovieLens datasets, we can

Show Name	Attributes	Popularity in Train
Training Items		
QI XL	formats/gamesandquizzes, genres/comedy, genres/entertainment, service/bbctwo	35286
Secrets of South America (1)	formats/documentaries, genres/factual, service/bbcthree	33239
The Great British Sewing Bee	genres/factual/beautyandstyle, genres/factual, service/bbctwo	37541
This World	formats/documentaries, genres/factual, genres/news, service/bbctwo	28385
Test Items		
QI (1)	formats/gamesandquizzes, genres/comedy, genres/entertainment, subjects/trivia, people/jack-dee, people/bill-bailey, people/alan-davies, people/stephen-fry, service/bbctwo	17867
QI (2)	formats/gamesandquizzes, genres/comedy, genres/entertainment, service/bbctwo	2164
Is Amanda Knox Guilty?	formats/documentaries, genres/factual, genres/factual/crimeandjustice, service/bbcthree	46643
Secrets of Bones	formats/documentaries, genres/factual, genres/factual/scienceandnature, genres/factual/scienceandnature/natureandenvironment, service/bbcfour	14502
Secrets of South America (2)	formats/documentaries, genres/factual, service/bbcthree	37013
Storyville	formats/documentaries, genres/factual, genres/factual/artsculturetheandthemedia, genres/factual/artscultureandthemedia/arts, service/bbcfour	3555
The Brits who Built the Modern World	formats/documentaries, genres/factual, genres/factual/artsculturetheandthemedia, genres/factual/artscultureandthemedia/arts, service/bbcfour	9804
The Truth about Webcam Girls	formats/documentaries, genres/factual, service/bbcthree	81237

Table 3.5: Training and test items of an iPlayer sample user.

conclude that the diversity of AN-P^3 is with few exceptions comparable or better than of the baseline methods at the same level of accuracy. Less useful is AN-P^3 for the BookCrossing dataset, since the maximal precision of AN-P^3 is lower than of the most precise baseline algorithms. But also for the BookCrossing data provides popularity normalization a clear increase in coverage, personalization and surprisal when compared to the corresponding metric score at the level of accuracy of the non-normalized method.

3.6 Example Recommendation Lists

In this section we illustrate the effect of popularity normalization of vertex transition probability-based item ranking with a set of recommendation lists for a sample iPlayer user. Table 3.5 lists the training and test items of the sample user from the iPlayer dataset. For the same user we generated recommendation lists with AN-P^3 and $\beta = 0.0$, 0.5 0.6, 0.7, 0.8, 0.9, 0.95 and 1.0 (see Table 3.6 for the top five recommendations of the lists). As expected, are the top places of the recommendation list for P^3 (same method as AN-P^3 with $\beta = 0.0$) exclusively populated with very popular TV shows. The first two recommendations of P^3 (*Top Gear* and *Outnumbered*) are the two most popular shows in the training set and the least popular show in the top five recommendations (*Dragons's Den*) was still the 13th most watched show during the training phase. Increasing β leads to a steady decrease of the total popularity of the top 5 items of the recommendation list and the list becomes dominated by items of very low popularity for $\beta > 0.9$ (*Select Committees* is the least popular show in the training set).

Rank	Show Name	Attributes	Popularity in Train
	$\beta = 0.0$ (AUC= 0.958, Prec@5 = 0.20, ILD@5 = 0.90, Surp@5 = 2.86, EILD@5 = 0.67, EFD@5 = 1.14)		
1	Top Gear	formats/magazinesandreviews, genres/factual, genres/entertainment, genres/factual/carsandmotors, service/bbctwo	132'462
2	Outnumbered	genres/comedy, genres/comedy/sitcoms, service/bbcone	147'674
3	The Truth about Webcam Girls	formats/documentaries, genres/factual, service/bbcthree	81'237
4	The Graham Norton Show	formats/discussionandtalk, genres/entertainment, service/bbcone	61'407
5	Dragons' Den	formats/reality, genres/entertainment, service/bbctwo	61'373
	$\beta = 0.5$ (AUC= 0.971, Prec@5 = 0.4, ILD@5 = 0.84, Surp@5 = 3.01, EILD@5 = 0.51, EFD@5 = 2.74)		
1	Top Gear	formats/magazinesandreviews, genres/factual, etc.	132'462
2	Secrets of South America (2)	formats/documentaries, genres/factual, service/bbcthree	37'013
3	The Truth about Webcam Girls	formats/documentaries, genres/factual, service/bbcthree	81'237
4	Outnumbered	genres/comedy, genres/comedy/sitcoms, service/bbcone	147'674
5	The Graham Norton Show	formats/discussionandtalk, genres/entertainment, service/bbcone	61'407
	$\beta = 0.6$ (AUC= 0.981, Prec@5 = 0.60, ILD@5 = 0.83, Surp@5 = 3.62, EILD@5 = 0.63, EFD@5 = 4.55)		
1	Secrets of South America (2)	formats/documentaries, genres/factual, service/bbcthree	37'013
2	QI (1)	formats/gamesandquizzes, genres/comedy, etc.	17'867
3	Top Gear	formats/magazinesandreviews, genres/factual, etc.	132'462
4	The Graham Norton Show	formats/discussionandtalk, genres/entertainment, etc.	61'407
5	The Truth about Webcam Girls	formats/documentaries, genres/factual, service/bbcthree	81'237
	$\beta = 0.7$ (AUC= 0.984, Prec@5 = 0.40, ILD@5 = 0.88, Surp@5 = 3.89, EILD@5 = 0.69, EFD@5 = 3.73)		
1	Secrets of South America (2)	formats/documentaries, genres/factual, service/bbcthree	37'013
2	QI (1)	formats/gamesandquizzes, genres/comedy, etc.	17'867
3	Top Gear	formats/magazinesandreviews, genres/factual, etc.	132'462
4	The Graham Norton Show	formats/discussionandtalk, genres/entertainment, etc.	61'407
5	Britain's Great War	formats/documentaries, genres/factual, genres/factual/history, service/bbcone	31'637
	$\beta = 0.8$ (AUC= 0.990, Prec@5 = 0.40, ILD@5 = 0.85, Surp@5 = 4.84, EILD@5 = 0.67, EFD@5 = 3.77)		
1	QI (1)	formats/gamesandquizzes, genres/comedy, etc.	17'867
2	Secrets of South America (2)	formats/documentaries, genres/factual, service/bbcthree	37'013
3	Have I Got Old News For You	formats/gamesandquizzes, genres/comedy/satire, genres/comedy, service/bbctwo	15'932
4	Horizon	formats/documentaries, genres/factual, genres/factual/scienceandnature, genres/factual/scienceandnature/scienceandtechnology, service/bbctwo	18'873
5	Britain's Great War	formats/documentaries, genres/factual, etc.	31'637
	$\beta = 0.9$ (AUC= 0.991, Prec@5 = 0.40, ILD@5 = 0.85, Surp@5 = 4.84, EILD@5 = 0.67, EFD@5 = 3.77)		
1	QI (1)	formats/gamesandquizzes, genres/comedy, etc.	17'867
2	Secrets of South America (2)	formats/documentaries, genres/factual, service/bbcthree	37'013
3	Have I Got Old News For You	formats/gamesandquizzes, genres/comedy/satire, etc.	15'932
4	Horizon	formats/documentaries, genres/factual, etc.	18'873
5	Britain's Great War	formats/documentaries, genres/factual, etc.	31'637
	$\beta = 0.95$ (AUC= 0.985, Prec@5 = 0.60, ILD@5 = 0.87, Surp@5 = 7.36, EILD@5 = 0.68, EFD@5 = 5.60)		
1	QI (1)	formats/gamesandquizzes, genres/comedy, etc.	17'867
2	Secrets of South America (2)	formats/documentaries, genres/factual, service/bbcthree	37'013
3	Asia Business Report	genres/news, service/bbcnews	45
4	QI (2)	formats/gamesandquizzes, genres/comedy, etc.	2'164
5	Have I Got Old News For You	formats/gamesandquizzes, genres/comedy/satire, etc.	15'932
	$\beta = 1.00$ (AUC= 0.961, Prec@5 = 0.20, ILD@5 = 0.81, Surp@5 = 12.6, EILD@5 = 0.70, EFD@5 = 1.57)		
1	Asia Business Report	service/bbcnews, genres/news	45
2	HARDtalk	formats/discussionandtalk, genres/news, service/bbcnews	36
3	QI (1)	formats/gamesandquizzes, genres/comedy, etc.	17'867
4	Select Committees	genres/factual, genres/news, genres/factual/politics, service/parliament	5
5	Eòrpa	genres/factual, genres/news, service/bbcalba	86

Table 3.6: Top 5 items and per-user metric scores of recommendation lists generated by AN-P^3 with values of β in the range $0.0 - 1.0$ for the user with the usage behavior presented in Table 3.5. Underlined items are hits in the test set of the user.

If we consider the user's four training items we could describe the user as predominantly interested in game-shows (*QI XL*) and factuals (*The Great British Sewing Bee*), especially documentaries (*Secrets of South America (1)* and *This World*). This preference profile is confirmed by the seven test items that also consist exclusively of game-shows and documentaries. We would expect a recommender system that maximizes for prediction accuracy to recommend shows from the same or similar genres. However, this is not the case for P^3, which, for example, also places a sitcom (*Outnumbered*) among the top 5 items of the recommendation list. AN-P^3 with $\beta = 0.8 - 0.9$, on the other hand, generates recommendation lists similar to the user's learned preference profile by placing only game-shows (*QI (1)* and *Have I Got Old News For You*) or documentaries (*Secrets of South America (2)*, *Horizon*, and *Britain's Great War*) on the top 5 places of the recommendation lists. In agreement with this observation the item attribute-based intra-list diversity score decreases and the prediction accuracy increases when compared to P^3. If we choose a value of $\beta > 0.9$ the top of the recommendation list reflects the user's preference profile again less accurately. In this example we include for example also a news show (*Asia Business Report*) at rank three of the recommendation list with $\beta = 0.95$. Since the iPlayer dataset contains many factual shows of very low popularity, for example recordings of parliament debates, recommendations generated with value for $\beta > 0.95$ seem to be of little use for most users.

Popularity normalized recommendation lists include low popularity items that are similar to the user's training items

3.7 Combining Popularity Normalization and Transition Probability Tuning

In Section 3.4 we showed that both optimization procedures, popularity normalization and transition probability tuning, can increase recommendation accuracy, coverage, personalization, and surprisal. In this section we describe the influence on recommendation performance when combining both methods. Since transition probabilities after random walks of length three combined with absolute popularity normalization (AN-) showed the best accuracy performance we limit our analysis to the AN-P^3_α algorithm. We measured the algorithm's performance for the MovieLens-M and iPlayer dataset with values of α ranging from -0.4 to 3.0 in steps of 0.2 and β ranging from -0.2 to 1.1 in steps of 0.1 and indicate the results in Figure 3.6 as contour lines in dependency of $\langle \alpha, \beta \rangle$ parameter pairs.

For both datasets and accuracy metrics (AUC and Prec@20) the measurements describe a similarly shaped surface with a distinct global maximum. When compared to the purely popularity normalized AN-P^3 ($\alpha = 1$) algorithm we found for AN-P^3_α no or only very small accuracy improvements. AN-P^3_α gives the best AUC performance for the MovieLens-M dataset at $\langle 1.0, 0.8 \rangle$ and for the iPlayer dataset at $\langle 0.4, 0.8 \rangle$, Prec@20 maxima were found at $\langle 0.8, 0.7 \rangle$ for MovieLens-M and at $\langle 0.4, 0.8 \rangle$ for iPlayer. The α values of AN-P^3_α that give the best accuracy performance are much smaller than for the non-popularity normalized ($\beta = 0$) method P^3_α (AUC and Prec@20 maxima of

No striking accuracy or diversity performance improvements for AN-P^3_α over AN-P^3

P^3_α were found for the MovieLens-M dataset at $\alpha = 1.8$ and for the iPlayer dataset at $\alpha = 1.6$). Hence, the accuracy improvements achievable with combined α and β tuning are not additive. In fact, the accuracy of the AN-P^3_α algorithm slightly increases for $\alpha < 1$, which is the opposite trend than for pure transition probability tuning with P^3_α.

As it is the case for accuracy, the surfaces indicating the diversity performance are roughly consistent between the datasets. The only exception is the ILD metric that indicates improving diversity for values of $\alpha > 1.0$ at $\beta = [0.6, 1.1]$ for MovieLens-M but decreasing diversity for values of $\alpha > 1.0$ in the same β range for the iPlayer data. As for accuracy, we did not find a parameter pair that gives clearly better performance in any of the four considered diversity dimension than AN-P^3. Our measurements suggest to set $\langle 0.8, 0.7 \rangle$ as default parameter pair in order to maximize recommendation list accuracy. A greater value of β would improve coverage, personalization, and surprisal but only on the expense of dramatically reduced accuracy, especially precision, and intra-list diversity. If we increase α slightly, maybe up to $\alpha = 1.2$, at constant $\beta = 0.7$, we can potentially improve surprisal, personalization, and coverage, causing a smaller loss in accuracy than for increasing β. The impact of the greater α value on intra-list diversity is unclear: our measurements suggest a notable increase for the MovieLens-M dataset but a slight decrease for the iPlayer dataset. However, combining α and β tuning does not seem to be a promising optimization approach if we consider the much larger parameter search space and the small achievable diversity improvements with an acceptable loss in accuracy.

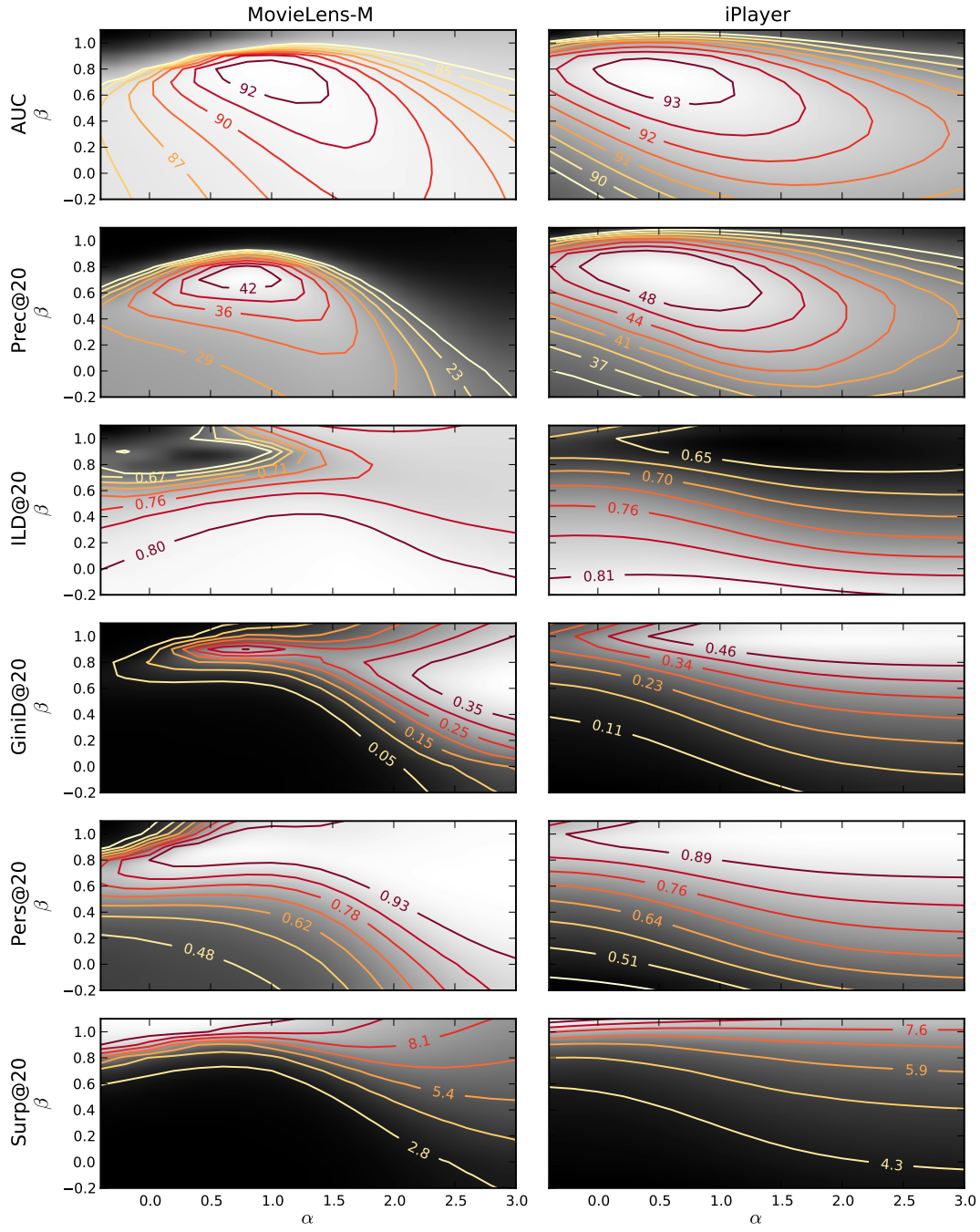


Figure 3.6: Contour plots of the accuracy and diversity performance of AN-P$^3_\alpha$ (T) in dependency of α in the range of $[-0.4, 3.0]$ and β in the range of $[-0.2, 1.1]$. Better recommender performance is indicated by a brighter background shade (brightness values for the plot areas between the measuring grid intersections were obtained with Gaussian interpolation).

4

Conclusions

Together with the emergence of large-scale graph storage and processing techniques over the last decade, also the interest of researchers in the field of recommender systems for graph-based approaches increased. Most of the various graph-based recommendation algorithms that have been proposed try to maximize recommendation accuracy, even-though it has been widely recognized that accuracy is not the only performance characteristic that needs to be considered when assessing the quality of a recommender system.

In this study we evaluated some of the most accurate graph-based collaborative filtering recommendation algorithms, i.e., P^3 and P^3_α, with an item ranking task on four implicit feedback datasets in terms of accuracy but also intra-list diversity, coverage, personalization, surprisal and rank sensitive and relevance aware versions of intra-list diversity and surprisal. The evaluation comprised the widely used MovieLens dataset, a dataset from a VOD system for TV shows and a dataset from an online bookstore. Besides a slightly worse accuracy of the proposed graph-based algorithms, we found a much worse performance for the most accurate graph-based methods in terms of coverage, personalization and surprisal when compared to recommenders using item-based nearest-neighbor classification or a latent factor model. The graph-based recommenders that achieved the best accuracy scores rank items according to the vertex transition probability of a random walk of length three (P^3 and P^3_α). However, the generated recommendation lists are dominated by the most popular items and are therefore of little use in helping users discover unknown items. The only graph-based recommendation strategy that showed good accuracy and non-accuracy performance ranks vertices by the entries in the Moore-Penrose pseudoinverse of the graph's Laplacian matrix (L^+). But this method is clearly disadvantageous compared to ranking vertices according to random walk transition probabilities from a computational point of view (L^+ can be obtained through a singular value decomposition of the Laplacian matrix and P^3, with lower computational effort, by raising the graphs transition probability matrix P to its third power).

In order to achieve better non-accuracy scores with vertex transition probability-based recommenders, we applied an item popularity dependent recommendation list re-ranking procedure, referred as popularity normalization. Popularity normalization changes the rank determining weight of an item (i.e., the vertex transition probability from the target user to the recommendation candidate items) in dependency of the

item's popularity and a parameter β. Increasing β reduces the weights of popular items and hence makes it more likely for items of low popularity to populate top ranks of the recommendation lists. With popularity normalization we observed not only better coverage, personalization and surprisal but also improved precision for small values of β. For greater values of β the accuracy of the recommender drops dramatically. We conclude that popularity normalized recommendation lists are more desirable from a user's perspective because they support users in discovering unknown items and from a provider's perspective since they contain also items from the long-tail. Furthermore popularity normalization allows to find an optimal trade-off between recommendation accuracy and the frequency of low popularity items in the recommendation list. Our experiments also showed that the relative performance of all evaluated recommenders is consistent between coverage, personalization and surprisal. For this reason it is thinkable to simplify the evaluation of recommender systems in future studies without losing explanatory power by measuring only one of these three performance dimensions. An even more straightforward methodology would be the use of a metric that combines the measurement of precision and long-tail item frequency, i.e., the rank sensitive and relevance aware EFD metric.

In this study we also compared the performance improvements achievable with popularity normalization against tuning of transition probabilities. Popularity normalization outperformed transition probability tuning significantly for all datasets in terms of accuracy. We also experimented with a recommendation algorithm that combines both optimization procedures but found no clear improvement for any accuracy or non-accuracy metric over the pure popularity normalization approach.

Many of the evaluated graph-based recommenders (e.g., P^3 and P^3_α) generated diverse lists of recommendations according to intra-list diversity, measuring the variety of meta attribute values among the recommended items. By introducing popularity normalization we observed a decrease of intra-list diversity while the accuracy increased. This means that the recommender generated recommendation lists with more similar items that better match the users' preference profile. It is worth noting that the very simple popularity-based recommendation strategy (MostPop) achieves better intra-list diversity scores than state-of-the-art recommendation algorithms. While it is possible that this finding could be confuted by experiments with different datasets (the most popular items in the used datasets feature apparently a wide variety of different item meta attributes), the usefulness of the intra-list diversity metric to assess recommendation quality seems to be questionable.

In our experiments we considered three different approaches to calculate vertex transition probabilities of short random walks: matrix algebra, random walk simulation, and probability tree traversal. The matrix algebra approach gave precise transition probability values and therefore slightly more accurate recommendations but is not applicable to real life datasets with millions of user feedbacks due to the high time and space complexity. Transition probabilities obtained by simulating random walks are estimates that converge to the actual values with increasing number of walks. By using this incremental calculation approach prediction accuracy can be traded in for runtime improvements, which can be desirable in a real world application. Calculating

vertex transition probabilities with a probability tree traversal gives precise probability values and circumvents the need to construct matrices that do not fit into the computers memory. With this approach the required computation time depends on the neighborhood size of the target user up to the considered random walk length and hence increases with increasing graph size. While the scalability of the simulation approach is better (computational complexity does not depend on the graph size given a fixed number of random walks per user) than of the tree traversal approach (computational complexity for a single user depends linearly on the users neighborhood size) the runtime of the latter for random walks of length three was short enough to apply the method to the biggest dataset of our evaluation. Therefore, we think that probability tree traversal is worth considering as an alternative implementation approach to random walk simulation when implementing a recommender based on vertex transition probabilities.

In the introduction to this work we mentioned that graph-based recommenders are a possible approach to overcome the flaw of limited coverage of state-of-the-art neighborhood-based recommendation methods thanks to the access to transitive relationship information encoded in the graph data structure. The results of our experiments do not support this hypothesis: the vertex ranking according to the transition probability for random walks of length three gave better recommendation precision than for random walks of length five, independent of the diameter of the underlying graph. Incorporating the preference profiles of users in four edges distance for the item ranking gave recommendation lists that are very similar to a global popularity ranking, which is a bad estimate of the actual preference profile of a particular user.

4.1 Future Work

The herein presented work bears opportunities for future research efforts in three different directions: additional off-line experiments, user studies, and applicability of the findings in real world systems.

In this work we studied the performance of graph-based recommenders in the implicit-feedback case only. We did not consider explicit rating values but this data could be introduced in the graph data structure as edge weights in order to, for example, influence the edge selection probability of random walks. This additional data would allow to better estimate a users preference profile and to predict item rating values. Hence, we could measure a recommender's accuracy not only based on the item ranking order but also by the deviation between the actual and predicted item rating value.

We proposed and evaluated two different re-ranking procedures that compensate for the impact of item popularity and improve precision of vertex transition probability-based recommenders (see Section 2.2). A future study could potentially discover more effective re-ranking procedures by further analyzing the distribution of ratings among items. It may also be worth testing the influence of popularity normalization on the performance of other graph- and non-graph-based recommenders.

Popularity normalization increases the probability to recommend items with fewer

ratings. Therefore, it seems reasonable that the method alleviates the item cold-start problem, which is immanent to collaborative filtering recommenders. To assert this assumption, we suggest to perform off-line experiments with datasets of different sparsity levels.

An interesting continuation of the presented work would be to test popularity normalized vertex transition probability-based recommenders (e.g., AN-P^3) in a user study to explore the effect of the re-ranking procedure on user satisfaction and perceived recommendation diversity. Beside the aim to detect the appropriate degree of popularity normalization, the result of the user study could help to decide for the best metric to estimate user-centric quality attributes in off-line experiments. It is reasonable to assume that not all users appreciate the same degree of popularity normalization, thus the procedure introduces a new dimension for recommendation personalization. From an application perspective we could think of a recommender system that analysis the rating history of a user to estimate the users preferred degree of popularity normalization. An other application could be an interactive recommender system that allow users to bias the recommendation generation process to more or less surprising recommendations by specifying the desired level of popularity normalization.

As mentioned in the introduction to this work, graph-based recommenders feature the advantageous property of a computationally cheap training phase. Recommendation generation, on the other hand, requires to run computationally expensive graph queries. The time required to generate a recommendation list for a single user with the AN-P^3 method using the largest training set by simulating random walks or performing a probability tree traversal was in the range of seconds on current commodity hardware. This computing time requirement makes it challenging to apply graph-based methods in a productive setting with thousand or more recommendation list requests per second. For this reason future research efforts should also attempt to reduce the response time of graph-based recommenders by developing efficient graph queries or recommendation list precalculation procedures.

Bibliography

Adomavicius, G. and Kwon, Y. (2011). Maximizing Aggregate Recommendation Diversity: A Graph-Theoretic Approach. In *Proceedings of the Workshop on Novelty and Diversity in Recommender Systems (DiveRS '11), at the 5th ACM International Conference on Recommender Systems (RecSys '11)*, pages 3–10, Chicago, IL, USA.

Adomavicius, G. and Kwon, Y. (2012). Improving Aggregate Recommendation Diversity Using Ranking-Based Techniques. *Knowledge and Data Engineering, IEEE Transactions on*, 24(5):896–911.

Aggarwal, C. C., Wolf, J. L., Wu, K.-L., and Yu, P. S. (1999). Horting Hatches an Egg: A New Graph-Theoretich Approach to Collaborative Filtering. In *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining (KDD '99)*, pages 201–212, San Diego, CA, USA.

Ahn, Y. A., Park, S., Lee, S., and Lee, S.-g. (2013). A Heterogeneous Graph-based Recommendation Simulator. In *Proceedings of the 7th ACM conference on Recommender systems (RecSys '13)*, pages 471–472, Hong Kong, China.

Baluja, S., Seth, R., Sivakumar, D., Jing, Y., Yagnik, J., Kumar, S., Ravichandran, D., and Aly, M. (2008). Video Suggestion and Discovery for YouTube: Taking Random Walks Through the View Graph. In *Proceeding of the 17th international conference on World Wide Web (WWW '08)*, pages 895–904, Beijing, China.

Bogers, T. (2010). Movie Recommendation using Random Walks over the Contextual Graph. In *Proceedings of the 2nd RecSys Workshop on Context-Aware Recommender Systems (CARS '10), at the 4th ACM International Conference on Recommender Systems (RecSys '10)*, Barcelona, Spain.

Cooper, C., Lee, S. H., Radzik, T., and Siantos, Y. (2014). Random Walks in Recommender Systems: Exact Computation and Simulations. In *Proceedings of the Companion Publication of the 23rd International Conference on World Wide Web Companion (WWW Companion '14)*, pages 811–816, Seoul, Korea.

Desrosiers, C. and Karypis, G. (2011). A Comprehensive Survey of Neighborhood-based Recommendation Methods. In Ricci, F., Rokach, L., Shapira, B., and Kantor,

P. B., editors, *Recommender Systems Handbook*, pages 107–144. Springer, Berlin, Germany.

Fouss, F., Pirotte, A., and Saerens, M. (2005). A Novel Way of Computing Similarities between Nodes of a Graph, with Application to Collaborative Recommendation. In *The 2005 IEEE/WIC/ACM International Conference on Web Intelligence (WI '05)*, pages 550–556, Halifax, NS, Canada.

Gantner, Z., Rendle, S., Freudenthaler, C., and Schmidt-Thieme, L. (2011). MyMediaLite: A Free Recommender System Library. In *Proceedings of the 5th ACM Conference on Recommender Systems (RecSys '11)*, pages 305–308, Chicago, IL, USA.

Gori, M. and Pucci, A. (2007). ItemRank: A Random-Walk Based Scoring Algorithm for Recommender Engines. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence (IJCAI '07)*, pages 2766–2771, Hyderabad, India.

Haveliwala, T. H. (2002). Topic-sensitive PageRank. In *Proceedings of the eleventh international conference on World Wide Web (WWW '02)*, pages 517–526, Honolulu, HI, USA.

Herlocker, J. L., Konstan, J. A., Terveen, L. G., and Riedl, J. T. (2004). Evaluating Collaborative Filtering Recommender Systems. *Information Systems, ACM Transactions on*, 22(1):5–53.

Huang, Z., Chen, H., and Zeng, D. (2004). Applying Associative Retrieval Techniques to Alleviate the Sparsity Problem in Collaborative Filtering. *Information Systems, ACM Transactions on*, 22(1):116–142.

Jamali, M. and Ester, M. (2009). TrustWalker: A RandomWalk Model for Combining Trust-based and Item-based Recommendation. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD '09)*, pages 397–405, Paris, France.

Koren, Y. and Bell, R. (2011). Advances in Collaborative Filtering. In Ricci, F., Rokach, L., Shapira, B., and Kantor, P. B., editors, *Recommender Systems Handbook*, pages 145–186. Springer, Berlin, Germany.

Lee, S., Park, S., Kahng, M., and Lee, S.-g. (2012). PathRank: A Novel Node Ranking Measure on a Heterogeneous Graph for Recommender Systems. In *Proceedings of the 21st ACM international conference on Information and knowledge management (CIKM '12)*, pages 1637–1641, Maui, HI, USA.

Malewicz, G., Austern, M. H., Bik, A. J., Dehnert, J. C., Horn, I., Leiser, N., and Czajkowski, G. (2010). Pregel: A System for Large-scale Graph Processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data (SIGMOD '10)*, pages 135–146, Indianapolis, IN, USA.

McNee, S. M., Riedl, J., and Konstan, J. A. (2006). Being Accurate is Not Enough: How Accuracy Metrics Have Hurt Recommender Systems. In *CHI '06 Extended Abstracts on Human Factors in Computing Systems (CHI EA '06)*, pages 1097–1101, Montreal, QC, Canada.

Page, L., Brin, S., Motwani, R., and Winograd, T. (1998). The PageRank Citation Ranking: Bringing Order to the Web. Technical report, Stanford University.

Rendle, S., Freudenthaler, C., Gantner, Z., and Schmidt-Thieme, L. (2009). BPR: Bayesian Personalized Ranking from Implicit Feedback. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence (UAI '09)*, pages 452–461, Montreal, QC, Canada.

Ricci, F., Rokach, L., and Shapira, B. (2011). Introduction to Recommender Systems Handbook. In Ricci, F., Rokach, L., Shapira, B., and Kantor, P. B., editors, *Recommender Systems Handbook*, pages 1–38. Springer, Berlin, Germany.

Shani, G., Chickering, M., and Meek, C. (2008). Mining Recommendations from the Web. In *Proceedings of the 2008 ACM Conference on Recommender Systems (RecSys '08)*, pages 35–42, Lausanne, Switzerland.

Shani, G. and Gunawardana, A. (2011). Evaluating Recommendation Systems. In Ricci, F., Rokach, L., Shapira, B., and Kantor, P. B., editors, *Recommender Systems Handbook*, pages 257–298. Springer, Berlin, Germany.

Smyth, B. and McClave, P. (2001). Similarity vs. Diversity. In *Proceedings of the 4th International Conference on Case-Based Reasoning: Case-Based Reasoning Research and Development (ICCBR '01)*, pages 347–361, Vancouver, BC, Canada.

Stutz, P., Bernstein, A., and Cohen, W. (2010). Signal/Collect: Graph Algorithms for the (Semantic) Web. In *Proceedings of the 9th International Semantic Web Conference on The Semantic Web - Volume Part I (ISWC '10)*, pages 764–780, Shanghai, China.

Vargas, S. and Castells, P. (2011). Rank and Relevance in Novelty and Diversity Metrics for Recommender Systems. In *Proceedings of the Fifth ACM Conference on Recommender Systems (RecSys '11)*, pages 109–116, Chicago, IL, USA.

Xiang, L., Yuan, Q., Zhao, S., Chen, L., Zhang, X., Yang, Q., and Sun, J. (2010). Temporal Recommendation on Graphs via Long- and Short-term Preference Fusion. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD '10)*, pages 723–731, Washington, DC, USA.

Zhang, Y. C., Séaghdha, D. O., Quercia, D., and Jambor, T. (2012). Auralist: Introducing Serendipity into Music Recommendation. In *Proceedings of the Fifth ACM International Conference on Web Search and Data Mining (WSDM '12)*, pages 13–22, Seattle, WA, USA.

Zhou, T., Kuscsik, Z., Liu, J.-G., Medo, M., Wakeling, J. R., and Zhang, Y.-C. (2010). Solving the apparent diversity-accuracy dilemma of recommender systems. *Proceedings of the National Academy of Sciences*, 107(10):4511–4515.

Ziegler, C.-N., McNee, S. M., Konstan, J. A., and Lausen, G. (2005). Improving Recommendation Lists Through Topic Diversification. In *Proceedings of the 14th International Conference on World Wide Web (WWW '05)*, pages 22–32, Chiba, Japan.

A

Appendix

A.1 Parameter Sets of Recommenders

In addition to the optimized parameters listed in Table A.1, we evaluated the recommenders with the following configurations. Recommenders not listed hereafter have no additional configuration options.

I-kNN and WI-kNN
Type of item similarity measure: cosine.

BPRMF
Number of gradient stochastic ascent iterations for training: 30, use item bias: false, length of iteration: 5, learning rate α: 0.05, regularization parameter for positive item factors: 0.0025, regularization parameter for negative item factors: 0.00025, regularization parameter for user factors: 0.0025.

P^3 (S) and P^5 (S)
See section 2.1 for a description of the simulation convergence criterion.

IR (S)
Decay or damping factor α: 0.85 (as proposed in Gori and Pucci [2007]), same simulation convergence criterion as described for P^3 (S) and P^5 (S) in Section 2.1 but with $n = 5$, $r = 20'000$ and a maximal number of iterations of 1'000'000 for all datasets.

IR (M)
Decay or damping factor α: 0.85, iterations per user: 20 (both parameter values were proposed in Gori and Pucci [2007]).

BnB
Edge weight: 0.5 (as proposed in Huang et al. [2004]), number of iterations: 35 (Huang et al. [2004] proposed 70 but 35 showed better accuracy).

OW (S) and CT (S)
Number of simulated random walks per user: fixed to $2 * |I|$, no convergence

criterion was used to control the simulation length. Each random walk had a destination vertex assigned and was terminated once this vertex was reached for OW, or once the start vertex was reached after the destination vertex was passed for CT. Hence each item vertex was reached or passed at least twice. In order to compensate for the small number of distinct walks and to improve the ranking quality, we also considered the step count of item vertices intermediately passed during the random walks.

A.2 Experiments Performed for Evaluation

Table A.1 lists the experiments performed to obtain the results presented in this work. The computing infrastructure used for the experiments consisted of a Slurm[1] cluster with 16 nodes. Each cluster machine has 128 GB of RAM and two Intel® Xeon® E5-2680V2 processors (25 MB Cache, 2.80 GHz base frequency) with 10 cores per processor (40 threads per machine).

A.3 P^3_α vs. AN-P^3 Significance Test

We verified the statistical significance of the performance differences between P^3_α (M/T) and AN-P^3 (T) with a two sided paired Wilcoxon signed rank test as proposed in Shani and Gunawardana [2011]. For each dataset we compared the performance of the two recommenders for the parameter values, α and β, that achieved the best AUC performance. We only considered metrics that can be calculated for each user independent of the recommendations for other users. The p-values of the Wilcoxon tests are listed in Table A.2. The following R[2] command was used to run the tests: `wilcox.test(<vector of` P^3_α `measurements>, <vector of AN-`P^3 `measurements>, mu=0, alt="two.sided", paired=T, conf.int=T, conf.level=0.999, exact=FALSE, correct=FALSE)`.

[1] Slurm Workload Manager - *slurm.schedmd.com*
[2] R software environment for statistical computing - *r-project.org*

	Recommender	Parameter	Dataset: MovieLens-S	MovieLens-M	iPlayer	BookCrossing
Baseline Methods	Perfect	N/A	+	+	+	+
	Random	N/A	+	+	+	+
	MostPop	N/A	+	+	+	+
	I-kNN	neighbourhood size (k)	10, 50, 100, 150, 200	10, 50, 100, 150, 200	10, 50, 100, 150, 200	10, 50, 100, 150, 200, 400, 800, 1600
	WI-kNN	neighbourhood size (k)	10, 50, 100, 150, 200	10, 50, 100, 150, 200	10, 50, 100, 150, 200	10, 50, 100, 150, 200, 400, 800, 1600, 3200
	BPRMF	number of latent factors	10, 50, 100, 150, 200	10, 50, 100, 150, 200	10, 50	10, 50, 100, 150, 200
Graph-based Methods	AN-P^3 (S)	β	-0.2 - 1.2 in 0.1 steps	-0.2 - 1.2 in 0.1 steps	-0.2 - 1.2 in 0.1 steps	-0.2 - 1.2 in 0.1 steps
	AN-P^3 (T)	β	-	-0.2 - 1.2 in 0.1 steps	-0.2 - 1.2 in 0.1 steps	-
	RN-P^3 (S)	β	-0.2 - 1.2 in 0.1 steps	-0.2 - 1.2 in 0.1 steps	-0.2 - 1.2 in 0.1 steps	-0.2 - 1.2 in 0.1 steps
	RN-P^3 (T)	β	-	-0.2 - 1.2 in 0.1 steps	-0.2 - 6.0 in 0.2 steps	-
	AN-P^5 (S)	β	-0.2 - 1.2 in 0.1 steps	-0.2 - 1.2 in 0.1 steps	-0.2 - 1.2 in 0.1 steps	-0.2 - 1.2 in 0.1 steps
	RN-P^5 (S)	β	-0.2 - 1.2 in 0.1 steps	-0.2 - 1.2 in 0.1 steps	-0.2 - 1.2 in 0.1 steps	-0.2 - 1.2 in 0.1 steps
	P^3_α (M)	α	-0.2 - 4.5 in 0.1 steps	-0.2 - 4.5 in 0.1 steps	x	-0.2 - 4.5 in 0.1 steps
	P^3_α (T)	α	-	-	-0.2 - 4.5 in 0.1 steps	-
	AN-P^3_α (T)	α, β	-	α: -0.4 - 3.0 in 0.2 steps, β: -0.2 - 1.1 in 0.1 steps	α: -0.4 - 3.0 in 0.2 steps, β: -0.2 - 1.1 in 0.1 steps	-
	P^5_α (M)	α	1.0	1.0	x	-0.2 - 4.5 in 0.1 steps
	3Path	N/A	+	+	+	+
	AN-IB-3Path	β	-0.2 - 1.2 in 0.1 steps	-0.2 - 1.2 in 0.1 steps	-0.2 - 1.2 in 0.1 steps	-0.2 - 1.2 in 0.1 steps
	IR (S)	N/A	+	x	+	x
	IR (M)	N/A	+	+	+	x
	BnB	N/A	+	+	x	+
	OW (S)	N/A	+	x	x	x
	CT (S)	N/A	+	x	x	x
	L^+ (M)	N/A	+	+	x	+

Table A.1: Experiments performed for evaluation. A '+' or a parameter specification denote experiments that were successfully performed. A 'x' indicates that the experiment failed due to insufficient memory of the compute node or that the experiment was aborted because the maximal available processing time of 24 hours expired. If we did not attempt to perform an experiment the table cell contains a '-'. Due to insufficient computing resources we could not measure the performance of the baseline method BPRMF with number of latent factors > 50 for the iPlayer dataset. Since the AUC accuracy of the kNN recommenders has not converged for a neighborhood size of 200 in the experiments with the BookCrossing dataset, we performed additional measurements for this dataset considering a larger neighborhood.

	MovieLens-S	MovieLens-M	iPlayer	BookCrossing
α (P^3_α (M/T))	1.7	1.8	1.5	0.9
β (AN-P^3 (T))	0.7	0.8	0.7	0.2
AUC	**<0.0001**	**<0.0001**	**<0.0001**	**<0.0001**
Prec@20	**<0.0001**	**<0.0001**	**<0.0001**	**<0.0001**
ILD@20	<u><0.0001</u>	<u><0.0001</u>	<u><0.0001</u>	N/A
Surp@20	**<0.0001**	**<0.0001**	**<0.0001**	**<0.0001**
EILD@20	<u>0.053</u>	<u><0.0001</u>	<u><0.0001</u>	N/A
EFD@20	**<0.0001**	**<0.0001**	**<0.0001**	**<0.0001**

Table A.2: P-values of Wilcoxon signed rank tests of P^3_α (M/T) versus AN-P^3 (T) at maximal level of AUC performance for per-user accuracy and diversity metrics. Figures in bold indicate a better average performance of AN-P^3, underlined figures a better performance of P^3_α.

A.4 Calculating Transition Probabilities with Signal/Collect

The Scala[3] code listings A.1, A.2, and A.3 exemplify random walk vertex transition probability calculations for a toy graph with the *Signal/Collect*[4] framework.

```
1  package com.signalcollect.examples
2
3  import com.signalcollect._
4  import com.signalcollect.configuration.ExecutionMode
5
6  class PsEdge[Id](t: Id) extends DefaultEdge(t) {
7
8    type Source = PsVertex[Id]
9    def signal = source.state / source.sumOfOutWeights * weight
10 }
11
12 class PsVertex[Id](id: Id) extends DataGraphVertex[Id, Double](id, 0.0) {
13
14   type Signal = Double
15   def collect: Double = signals.sum
16 }
17
18 object Ps extends App {
19   val g = new GraphBuilder[Any, Any]().build
20   g.awaitIdle
21
22   var user1 = new PsVertex(1)
23   g.addVertex(user1)
24   user1.state = 1.0
25   g.addVertex(new PsVertex(2))
26   g.addVertex(new PsVertex(3))
27   g.addVertex(new PsVertex(4))
28   g.addVertex(new PsVertex(5))
29   g.addVertex(new PsVertex(6))
30
31   g.addVertex(new PsVertex('A'))
32   g.addVertex(new PsVertex('B'))
33   g.addVertex(new PsVertex('C'))
34   g.addVertex(new PsVertex('D'))
35
36   g.addEdge(1, new PsEdge('A'))
37   g.addEdge('A', new PsEdge(1))
38
39   g.addEdge(2, new PsEdge('A'))
40   g.addEdge(2, new PsEdge('B'))
41   g.addEdge('A', new PsEdge(2))
42   g.addEdge('B', new PsEdge(2))
43
44   g.addEdge(3, new PsEdge('A'))
45   g.addEdge(3, new PsEdge('B'))
46   g.addEdge(3, new PsEdge('C'))
47   g.addEdge('A', new PsEdge(3))
48   g.addEdge('B', new PsEdge(3))
49   g.addEdge('C', new PsEdge(3))
50
51   g.addEdge(4, new PsEdge('B'))
52   g.addEdge(4, new PsEdge('C'))
53   g.addEdge('B', new PsEdge(4))
54   g.addEdge('C', new PsEdge(4))
55
56   g.addEdge(5, new PsEdge('C'))
57   g.addEdge(5, new PsEdge('D'))
58   g.addEdge('C', new PsEdge(5))
59   g.addEdge('D', new PsEdge(5))
60
61   g.addEdge(6, new PsEdge('D'))
62   g.addEdge('D', new PsEdge(6))
63
64   g.awaitIdle
65   val stats = g.execute(ExecutionConfiguration.withExecutionMode(ExecutionMode.Synchronous).
       withStepsLimit(3))
66   println(stats)
67
68   g.foreachVertex{v =>
69     println("Transition␣Probability␣from␣user␣1␣to␣vertex␣with␣id␣" + v.id + ":␣" + v.state)
70   }
71
72   g.shutdown
73 }
```

Listing A.1: Calculating transition probabilities for user 1 after a three step random walk with $\alpha = 1.0$ (P^3).

[3] Scala programming language - *scala-lang.org*

[4] Signal/Collect graph processing framework - *uzh.github.io/signal-collect*

```
1  package com.signalcollect.examples
2
3  import com.signalcollect._
4  import com.signalcollect.configuration.ExecutionMode
5
6  class PsAlphaEdge[Id](t: Id) extends DefaultEdge(t) {
7
8    type Source = PsAlphaVertex[Id]
9    def signal = source.state.map{_ / source.sumOfOutWeights * weight}
10 }
11
12 class PsAlphaVertex[Id](id: Id) extends DataGraphVertex[Id, List[Double]](id, List()) {
13
14   type Signal = List[Double]
15   def collect: List[Double] = signals.flatten.toList
16 }
17
18 class PsAlphaAggregator[IdType] (alpha: Double) extends ModularAggregationOperation[Map[IdType,
       List[Double]]]{
19   val neutralElement = Map[IdType,  List[Double]]()
20   def extract(v: Vertex[_, _]): Map[IdType,  List[Double]] = {
21     try {
22       Map[IdType,  List[Double]]((v.id.asInstanceOf[IdType], List(v.state.asInstanceOf[List[
             Double]].foldLeft(0.0)(_+ math.pow(_, alpha)))))
23     }
24   }
25   def aggregate(a: Map[IdType, List[Double]], b: Map[IdType, List[Double]]): Map[IdType, List[
         Double]] = a ++ b
26 }
27
28 object PsAlpha extends App {
29   val g = new GraphBuilder[Any, Any]().build
30   g.awaitIdle
31
32   var user1 = new PsAlphaVertex(1)
33   g.addVertex(user1)
34   user1.state = List(1.0)
35   g.addVertex(new PsAlphaVertex(2))
36   g.addVertex(new PsAlphaVertex(3))
37   g.addVertex(new PsAlphaVertex(4))
38   g.addVertex(new PsAlphaVertex(5))
39   g.addVertex(new PsAlphaVertex(6))
40
41   g.addVertex(new PsAlphaVertex('A'))
42   g.addVertex(new PsAlphaVertex('B'))
43   g.addVertex(new PsAlphaVertex('C'))
44   g.addVertex(new PsAlphaVertex('D'))
45
46   g.addEdge(1, new PsAlphaEdge('A'))
47   g.addEdge('A', new PsAlphaEdge(1))
48
49   g.addEdge(2, new PsAlphaEdge('A'))
50   g.addEdge(2, new PsAlphaEdge('B'))
51   g.addEdge('A', new PsAlphaEdge(2))
52   g.addEdge('B', new PsAlphaEdge(2))
53
54   g.addEdge(3, new PsAlphaEdge('A'))
55   g.addEdge(3, new PsAlphaEdge('B'))
56   g.addEdge(3, new PsAlphaEdge('C'))
57   g.addEdge('A', new PsAlphaEdge(3))
58   g.addEdge('B', new PsAlphaEdge(3))
59   g.addEdge('C', new PsAlphaEdge(3))
60
61   g.addEdge(4, new PsAlphaEdge('B'))
62   g.addEdge(4, new PsAlphaEdge('C'))
63   g.addEdge('B', new PsAlphaEdge(4))
64   g.addEdge('C', new PsAlphaEdge(4))
65
66   g.addEdge(5, new PsAlphaEdge('C'))
67   g.addEdge(5, new PsAlphaEdge('D'))
68   g.addEdge('C', new PsAlphaEdge(5))
69   g.addEdge('D', new PsAlphaEdge(5))
70
71   g.addEdge(6, new PsAlphaEdge('D'))
72   g.addEdge('D', new PsAlphaEdge(6))
73
74   g.awaitIdle
75   val stats = g.execute(ExecutionConfiguration.withExecutionMode(ExecutionMode.Synchronous).
         withStepsLimit(3))
76   println(stats)
77
78   var alpha = 1.5
79   var m = g.aggregate(new PsAlphaAggregator[Any](alpha))
80   var sum = m.values.flatten.sum
81
82   g.foreachVertex{v =>
83     println("Transition␣Probability␣from␣user␣1␣to␣vertex␣with␣id␣" + v.id + "␣with␣alpha=" +
           alpha +":␣" + m.get(v.id).get.head / sum)
84   }
85
86   g.shutdown
87 }
```

Listing A.2: Calculating normalized transition probabilities for user 1 after a three step random walk with $\alpha = 1.5$ (P^3_α).

```
1  package com.signalcollect.examples
2
3  import com.signalcollect._
4  import com.signalcollect.configuration.ExecutionMode
5
6  class PsBEdge[Id](t: Id) extends DefaultEdge(t) {
7
8    type Source = PsBVertex[Id]
9    def signal = source.state.mapValues { _/ source.sumOfOutWeights * weight}
10 }
11
12 class PsBVertex[Id](id: Id) extends DataGraphVertex[Id, Map[Id, Double]](id, Map[Id, Double]()) {
13
14   type Signal = Map[Id, Double]
15   def collect: Map[Id, Double] = signals.flatten.groupBy(_._1).mapValues(_.map(_._2).sum)
16 }
17
18 object PsB extends App {
19   val g = new GraphBuilder[Any, Any]().build
20   g.awaitIdle
21
22   for (id  <- 1 to 6) {
23     var user = new PsBVertex(id)
24     user.setState(Map(id -> 1.0))
25     g.addVertex(user)
26   }
27
28   g.addVertex(new PsBVertex('A'))
29   g.addVertex(new PsBVertex('B'))
30   g.addVertex(new PsBVertex('C'))
31   g.addVertex(new PsBVertex('D'))
32
33   g.addEdge(1, new PsBEdge('A'))
34   g.addEdge('A', new PsBEdge(1))
35
36   g.addEdge(2, new PsBEdge('A'))
37   g.addEdge(2, new PsBEdge('B'))
38   g.addEdge('A', new PsBEdge(2))
39   g.addEdge('B', new PsBEdge(2))
40
41   g.addEdge(3, new PsBEdge('A'))
42   g.addEdge(3, new PsBEdge('B'))
43   g.addEdge(3, new PsBEdge('C'))
44   g.addEdge('A', new PsBEdge(3))
45   g.addEdge('B', new PsBEdge(3))
46   g.addEdge('C', new PsBEdge(3))
47
48   g.addEdge(4, new PsBEdge('B'))
49   g.addEdge(4, new PsBEdge('C'))
50   g.addEdge('B', new PsBEdge(4))
51   g.addEdge('C', new PsBEdge(4))
52
53   g.addEdge(5, new PsBEdge('C'))
54   g.addEdge(5, new PsBEdge('D'))
55   g.addEdge('C', new PsBEdge(5))
56   g.addEdge('D', new PsBEdge(5))
57
58   g.addEdge(6, new PsBEdge('D'))
59   g.addEdge('D', new PsBEdge(6))
60
61   g.awaitIdle
62   val stats = g.execute(ExecutionConfiguration.withExecutionMode(ExecutionMode.Synchronous).
         withStepsLimit(3))
63   println(stats)
64
65   g.foreachVertex { v =>
66     v.state.asInstanceOf[Map[Any, Double]].foreach{e =>
67       println("Transition␣Probability␣from␣user␣" + e._1 + "␣to␣item␣" + v.id + ":␣" + e._2)
68     }
69   }
70   g.shutdown
71 }
```

Listing A.3: Calculating transition probabilities for all users after a three step random walk with $\alpha = 1.0$ (P^3).

List of Figures

List of Tables