Recommending Long-Tail Items Using Extended Tripartite Graphs

Andrew Luke
Computer Science Department
Brigham Young University
Provo, Utah 84602, USA
kiyotaka.a.l@gmail.com

Joseph Johnson
Computer Science Department
Brigham Young University
Provo, Utah 84602, USA
josephjohnson11@gmail.com

Yiu-Kai Ng
Computer Science Department
Brigham Young University
Provo, Utah 84602, USA
ng@compsci.byu.edu

***Abstract*—With the popular and increasing power of the Internet these days, the effort of distributing and inventory costs of stocking various online retailing items are nearly negligible. In addition to selling popular, called "short-head", items in large quantities, online retailers, such as Amazon, offer a large number of unique items, called "long tail", with relatively small quantities sold. Retailers realize that it has high value to sell items from the long-tail category, since for users these long-tail items could meet the interest of them and surprise them simultaneously. Retailers also recognize that long-tail items can be an untapped source of revenue for a business; however, it is difficult to connect customers with long-tail items they are interested in, since they are unaware of them. Recommender systems help bridge the gap between users and long-tail items by learning user preferences and recommending appropriate items to them. In this paper, we propose a new tripartite graph recommender system, which is designed to suggest long-tail items. Compared with other graph-based recommender systems, our proposed recommendation system solves the tripartite variant problem suffered by existing approaches for having a low diversity score. A rework of the tripartite graph system is introduced, called the *extended tripartite graph system*, which enhances the performance of existing long-tail recommendation approaches measured by using two widely-used performance metrics: recall and diversity. Experimental results on the extended tripartite graph algorithm verify its merits and novelty.**

***Keywords*-long-tail items; recommendation; tripartite graphs**

I. Introduction

In an inventory, items can be generally divided into short-head items and long-tail items. *Short-head* items are items that are well-known and popular to users, whereas *long-tail* items are products that are lesser-known. It is not the case that long-tail items are inferior to short-head items, i.e., these items may still be favorably reviewed and well-received, but they are niche and relatively unknown to the general population, especially potential buyers.

To businesses, long-tail items are of great interest as they have the potential to be a significant source of revenue, whereas short-head items are generally more competitive, since popular items are well-known and many businesses carry the products which drives profits per item down. The lower popularity of long-tail items means that fewer competitors carry the items, allowing businesses to make more profit off of the items. Carrying a niche product can also draw new customers—people are attracted to a business that consistently offers interested, hard-to-find items.

Knowing who is interested in long-tail items, however, is a difficult problem to solve [11]. There are many long-tail items and since they are niche products, they are not visible, relatively unfamiliar, and unexplored to a given customer. Due to the low popularity of a long-tail item with which data about the item is sparse, purchasing and preference information on the item are often missing. The less rating and purchase information is available about an item, the more difficult it is for a recommender to correctly learn about the item in order to make good recommendations.

In the past, Yin et al. [15] and Shang et al. [13] have proposed different algorithms that enhance long tail recommendations. Yin et al. argue that traditionally, the Pareto Rule, or the 80/20 rule, seemed to hold in that a large amount (80%) of company revenues was generated by relatively few (20%) products. Yin's method modifies the probability variables of the hitting time algorithm to increase the likelihood that a random walker reaches the long tail regions of a bipartite graph, whereas Shang et al.'s approach is a collaborative filtering method that employs a tripartite graph and random walkers to find users with similar tastes as a user u [1]. Our proposed method combines elements of these methods, as well as elements from the familiar PageRank algorithm, to increase the likelihood that u is recommended items (i) whose characteristics are appealing to u, and (ii) are found in the long tail region. Specifically, we reduce the long tail recommendation problem to traversing a tripartite graph through a Markov process. In other words, we represent the tripartite graph as a stochastic matrix and after t ($\geq$ 1) iterations make recommendations to u based on the probability of arriving at each of the items available.

In this paper, we look at the long-term recommender system developed by Johnson et al. [7], which has been shown to outperform Yin's and Shang's approaches, and rework it better to recommend long-tail items. We first discuss work relevant to Johnson et al.'s tripartite graph recommender system. Hereafter, two test metrics, *recall* and *diversity*, are defined. A proposal of a rework of the tripartite graph recommender algorithm is given, along with

the rationale for changes and performance results for recall and diversity. The results are compared against the original algorithm and previous long-tail recommender systems.

The rest of the paper is organized as follows. In Section II, we present existing representative recommender systems that target the long-tail problem. In Section III, we introduce our extended tripartite graph algorithm used by our recommendation system. In Section IV, we evaluate the performance of our recommendation approach. In Section V, we give a concluding remark.

II. Related Work

Recommendation systems have typically been divided into two categories: *content-based* filtering and *collaborative-based* filtering. Collaborative-based filtering provides recommendations based on the users most similar to a user u or the products most similar to the products p rated by u [5], [10]. The weakness of this process is that it tends to provide local, trivial recommendations. Content-based filtering refers to predicting a rating by u for a product p based on a feature vector of descriptions of products previously rated by u [12]. Unfortunately, most products lack elaborate descriptions, which makes for poor predictions. In addition, products that do not have similar features to those rated by u are not recommended. In other words, this method provides no novel recommendations. The usage longevity of collaborative-based filtering and content-based filtering is that they tend to do well in the head market. However, as mentioned before, we believe the real value to users comes from long-tail recommendations; the introduction of products they would likely never discover through a non-algorithmic search.

Instead of relying on collaborative-based or content-based filtering approach, Yin et al. [15] introduce a number of bipartite graph-based recommender algorithms in which users and items are nodes connected based on users' ratings of items. In their hitting time (denoted HT) algorithm, given an adjacency matrix A of nodes which represent users and items and edges which represent ratings, edges in the graph are defined as $P_{i,j} = \frac{a(i,j)}{d_i}$, where $P_{i,j}$ is the *weight* of the edge between user i and item j, $a(i,j)$ is a *user rating* connecting an item j and a user i as contained in A, and $d_i = \sum_{k=1}^{n} a(i,k)$ is the degree of user i.

The hitting time, as defined in [15], from (node) j to (node) q, denoted $H(q|j)$, is the expected number of steps that a random walker starting from item j ($j \neq q$) will take to reach user q. Hitting time has the property such that it is *lower* when q and j are relevant and few users have rated j. Two nodes are *relevant* when (i) there are many paths that connect them, (ii) the paths are short, and (iii) the edges in the path have a high weight, i.e., a user rated that item highly. This is ideal in trying to make long-tail recommendations, i.e., given a user and an item, the hitting time score will be minimized if they are relevant and the item is a long-tail item. We can calculate a hitting time score for every user for every item. This gives a list of recommendations for a user, along with how strongly an item is being recommended.

Yin et al. also proposes two other graph-based algorithms, which are referred as the $AC1$ and $AC2$ algorithms [15]. Essentially, the AC algorithms build on their HT algorithm by weighting some user's ratings more than others. Users who have limited preferences are considered "experts" and are given more consideration than people who has a wide range of tastes. (The formal definitions of the AC algorithms are given in Section IV-A2.)

Shang et al. [13] present a tripartite graph-based approach to solve the long-tail recommendation problem. They supplement user-item data with user-tag data. Connections between users and items are a binary value, regardless of the user's rating, i.e., if a user rated an item, then that connection has a weight of 1, and 0 otherwise. Their approach is most similar to a collaborative filtering system, where users are compared to other users, and items are recommended if they are rated highly by similar users.

III. Our Tripartite Recommendation System

In discussing graph structure, we refer to individual users, items, and other latent information as "nodes". Each node is contained within a "set", which is a group of similar elements—users, items, or latent information. The edge that connects nodes is referred to as a "connection" or "path", and the strength of the connection as its *weight*.

In constructing the graph structure to capture the relationships among users, items, and latent information, Joseph et al.'s tripartite recommender system [7] is an extension of Yin's graph-based recommendation approach [15]. Yin's bipartite approach uses only item and user data, and Johnson's tripartite approach adds in latent information in a new set which is connected to the item and user sets. It is anticipated that the latent information helps fill in gaps where user-item data is sparse and serves a bridge which provides another path when traversing the graph. Recall that in Yin's hitting time algorithm, the more short paths connect a user to an item, the stronger the recommendation of the item for that user. In the case of the tripartite algorithm, the bipartite graph is supplemented by latent information, such as *genre*. The graphical representation shown in Figure 1 illustrates the basic differences in structure between Yin's bipartite approach and Johnson's tripartite method. Note that while the tripartite algorithm of [7] uses only genre information specifically as examples, the algorithm can be used on any kind of latent information that is set based.

Regarding movie's genre, the tripartite graph in [7] uses the *full* genre, which is essentially a list of genre tags such as "Action, Adventure, Comedy, Romance". Each node in the genre set is a set of full genres represented in the item set (meaning every item node has its full genre represented in the genre set). The tripartite graph in [7] uses the same

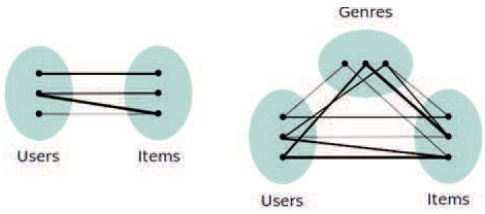

Figure 1. Yin et al.'s Bipartite [15] (on the left) versus Johnson et al.'s Tripartite [7] Graph Structure (on the right)

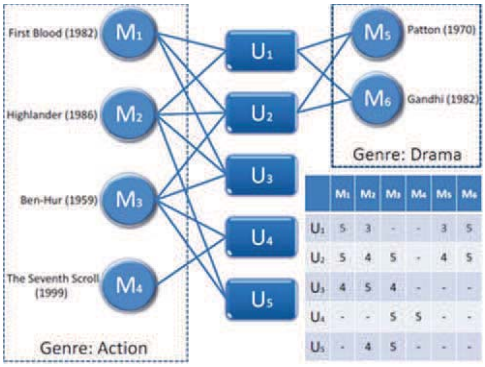

Figure 2. Bipartite representation of user-item dataset in [15]

algorithm as Yin et al.'s hitting time algorithm [15] discussed earlier. User-item relations are still defined by the rating a user gives an item. However, the hitting time algorithm does not define the weights of user-genre connections or genre-item connections. Instead, they are defined as follows based on the movie genres (see Figure 2 for an example):

- *User-genre*: the user's average rating for movies they have seen with the given genre such that 0 is given if the user has not rated any movies with this genre.
- *Genre-item*: the average rating for the item across all users if the genre matches the item, and 0 otherwise.

Johnson et al.'s tripartite graph algorithm [7] also uses a Markov process [4] to simulate random-walker movements (as seen in Yin's hitting time algorithm [15]). The Markov process starts with a stochastic transition matrix, which holds the probabilities of a random walker traveling from one node to another. If there is only one edge from node A to node B, then the probability that the random walker moves from A to B is 1.0. The *more* edges a node has, the *less likely* it is for a random walker to travel from that node to a specific node. By multiplying this stochastic transition matrix to itself, we can obtain the probability of a random walker starting at a node X ending up on node Y in two steps. By continuing to multiply this matrix by the transition matrix, we can calculate the probabilities of a random walker starting on node X and ending on node Y in T ($\geq$ 1) number of steps. Essentially, this T value dictates how long the random-walker is allowed to traverse the graph.

Johnson et al. argue that *lower* values of T are *more appropriate* for long-tail item recommendation, since the higher the T-value, the more likely a random walker is to end up on a short-head node. Intuitively, if a random walker were to have a very *long* time to explore a graph, it is mostly likely to end up on nodes with a higher degree, giving the walker a *short* period of time to crawl the graph means that it will be more likely to end on nodes that are relevant to its starting node. By keeping values of T *low*, the high degree of long-tail items has less effect on the recommendation strength. Johnson et al. [7] calculated recall and diversity scores for T = [3, 5, 7]. We consider when T = [3, 5], which are the $T3$ and $T5$ tripartite graph algorithms in [7], since they had better experimental results than $T7$.

The MovieLens dataset was used to generate *recall* and *diversity* scores for the Johnson et al.'s tripartite algorithm [7], which are formally defined below. The dataset contains information on a set of *users* and *movies*, along with a list of *ratings* created by the users for the movies.

A. Testing Metrics

Recall and *diversity* are the two metrics used to evaluate Johnson et al.'s tripartite recommender system. Recall measures a recommender system's ability to correctly learn a user's preferences, which is calculated as follows:

1) A set of users is chosen randomly.
2) For each user, an item is randomly chosen from the set of items rated 5 stars by the user.
3) Rating information relating the item to the user is scrubbed from the data set, and the scrubbed data set contains no information that the user ever rated the particular item.
4) For each user, a set of items unrated by the user is randomly selected. The number of selected items is the same for all users.
5) The recommendation algorithm is run using the scrubbed data set.
6) The recommendation score is created for the items in Steps 2 and 4, which are sorted according to the scores.
7) A $Recall@N$ measure is taken for various values of N ($\geq$ 1). $Recall@5$ ($Recall@10$, respectively) checks whether the 5-star item (from Step 2) is in the top-5 (top-10, respectively) recommendations, and so on. This is repeated for each user.
8) An overall *Recall* value is calculated for each value of N. $Recall@N$ is defined as

$$Recall@N = \frac{\sum Hit@N}{|L|} \quad (1)$$

where $Hit@N$ = 1 if the item in Step 2 is in the top-N items and 0 otherwise, and L is the number of users chosen in Step 1.

In *Recall*, a user's highly-rated item suggests a preference in the user's tastes. The recommender should learn those tastes, and make recommendations based on that knowledge. While we know that the user likes the item chosen in Step 2, the recommender does not have access to that information (because the data was scrubbed) and must learn it. If the recommender is learning the user's preferences correctly, it should give a strong recommendation for the item chosen in Step 2, resulting in a higher recall value.

The *diversity* score [3], on the other hand, measures how diverse a set of items a recommender suggests. Intuitively, if a recommendation algorithm suggests a very narrow set of items to a wide variety of users, it's likely that the suggestions are not niche, long-tail items but rather the generally popular short-head items. A higher diversity score indicates that the recommender system is suggesting a variety of items across many users that are likely preferred by the users. The *diversity* score is computed as

$$Diversity = \frac{\cup_{u \in U} R_u}{|I|} \tag{2}$$

where U is a set of users, R_u is the set of items recommended for user u, and $|I|$ is the maximum possible number of unique items that could be recommended. Specifics about the calculation of *recall* and *diversity* for our tripartite graph recommender algorithm are discussed in Section IV-B.

B. Recalculating Diversity

The diversity score, as defined in Equation 2, is used by Johnson et al.'s recommendation system [7]. In their tripartite graph system, U is defined as 200 (i.e., 200 random users), R_u is the top-10 items recommended to user u, and $|I|$, which is the maximum number of unique items that could be recommended, is set to be 200×10 = 2,000 items.

The original tripartite graph recommender as proposed by Johnson et al. makes recommendations for items that users had already rated. For example, if a user gave the rating of 5 to a movie, then it tends to show up as a highly recommended item for that user. In a practical recommender system, it would not make sense to recommend items that a user has already seen and rated, so the original algorithm was changed to remove these unnecessary recommendations in our new tripartite graph algorithm.

C. Our Enhanced Tripartite Graph Algorithm

We have improved the tripartite algorithm developed by Johnson et al. [7], and we refer ours as the *extended tripartite recommender*. The extended algorithm connects the user, genre, and item sets according to following setups:

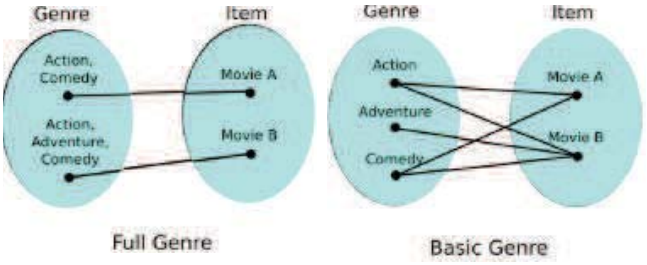

Figure 3. Full versus basic genre graphs

1) *User-item*: the user's rating for the item is 0 if the user has not rated the item.[1]
2) *User-genre*: the Bayesian average for the genre as defined in Equation 4.
3) *Genre-item*: the *average* rating of the item divided by the number of basic genres for applicable basic genres.

In making changes to Johnson et al.'s algorithm, we seek to change the user-genre relationship to better utilize the data, and the genre-item relationship to create more paths in the graph. The extended system uses "basic genres", a split-up version of the "full genre" used by Johnson et al.'s tripartite graph algorithm. Basically, a *full genre* consists of a set of tags applied to a movie, whereas a *basic genre* breaks up the full genre into its individual tags. For example, "Action, Adventure, Comedy" is a full genre, whereas its basic genres are "Action," "Adventure," and "Comedy."

The original tripartite algorithm proposed by Johnson et al. [7] does *not* fully take advantage of this connectedness metric when connecting an item node to a genre node, i.e., there is always one and only one path from an item node to a genre node, as an item only has one matching full genre. We have observed that creating *more* paths allows the algorithm to make *better* recommendations. To improve the connectivity, we move from using a full genre to using basic genres. In the extended tripartite algorithm, we change the nodes in the genres set to the set of individual basic genres, as opposed to a set of full genres so that an item can have multiple connections to the genres node based on the number of basic genres it has (see Figure 3 for an example).

1) Genre-Item: A feature of graph-based recommenders is the ability to make recommendations based on the *degree of connection* between a user and an item, which depends on the weight of a path and the number of paths. The *weight* of a connection in our extended tripartite algorithm is defined as

$$W_{i,bg} = \begin{cases} \frac{i_{avg}}{|BG_i|}, & bg \in BG_i \\ 0, & bg \notin BG_i \end{cases} \tag{3}$$

where i is an item, i_{avg} is the average rating for i, bg is a basic genre, and BG_i is the set of basic genres applied to i.

[1]Note that this is the same in Johnson et al.'s algorithm.

We divide the average rating by the number of basic genres to avoid giving items with many basic genres too much weight. Consider two movies with the same average rating, but the first movie has one basic genre, and the second has ten. If we do not divide the average rating by the number of basic genres, there is one path from the first movie to the genre set, but ten equally strong paths from the second movie to the genre set. This would cause the recommender to lean towards recommending the second item higher because it has many strong paths. Dividing the average rating by the number of basic genres means that items with fewer basic genres will have a few *strong* paths, and items with many basic genres will have many *weaker* paths, and the recommender avoids a bias towards one over the others.

2) User-Genre: In Johnson et al.'s algorithm [7], the weight of a user-genre relationship is the *average* rating of a genre across the movies a user has seen, but there is some *loss* of information in taking the simple average. Consider the case where a user likes western movies, and has rated 10 western movies, rating them an average of 4.7 out of 5. The user has also seen one sci-fi movie, and they rated it a 5. Taking the simple average would cause the recommender system to lean more towards recommending sci-fi movies, since they have a higher average rating. However, recommending a western movie would be more in line with the user's preference. This suggests that the connection between a user and a genre should be weighted more towards genres the user has rated many times. In improving the original tripartite graph recommender system, we use a *Bayesian average* [14] in weighting the user-genre relationship.

$W_{u,bg}$ =

$$\frac{avg\_votes_u \times avg\_rating_u + votes_{u,bg} \times rating_{u,bg}}{avg\_votes_u + votes_{u,bg}} \tag{4}$$

where u is a user, bg is a basic genre, avg_votes_u is the average number of items u has rated per basic genre, which is computed by summing the number of items a user has rated that has a given basic genre and taken an average of that value across all basic genres, avg_rating_u is u's average rating of basic genres, $votes_{u,bg}$ is the number of ratings u has made for bg, and $rating_{u,bg}$ is the average rating u has given bg.

The Bayesian average has the desirable property that as a user rates more items of a genre, the weighting of the Bayesian average shifts to the true average for that genre. Based on this strategy, we avoid the scenario when a genre with few ratings but a high average rating is recommended ahead of a genre with many ratings but a lower average.

Note again that while the extended tripartite system uses genre information specifically throughout the discussion, the algorithm works for any set-based latent information.

D. Using Latent Semantic Attributes on Long-Tail Items

Latent Semantic Indexing (LSI) [6] is a mathematical method used to determine the relationship between terms and concepts in content. It has been shown that we can improve the accuracy of many information retrieval applications based on latent semantic analysis. Given a term-item (i.e., attribute-item) frequency matrix, LSI can be used to decompose the matrix into two matrices of reduced dimensions and a diagonal matrix of singular values, which can be used in solving the long-tail problem. A latent factor yields a dimension in the reduced space which represents groups of highly correlated semantic terms. Reducing the dimensionality of the original matrix reduces the amount of noise in the data, as well as its sparsity, and as a result improving retrieval based on the computation of similarities among different items. We apply this idea to create a reduced dimension space for the semantic attributes associated with long-tail items.

Singular Value Decomposition (SVD) [8] is a well-known technique used in LSI to perform matrix decomposition. We apply SVD on the semantic attribute matrix $S_{n\times d}$ by decomposing it into three matrices, i.e.,

$$S_{n\times d} = U_{n\times r} \bullet \sum\nolimits_{r\times r} \bullet V_{r\times d} \tag{5}$$

where U and V are two *orthogonal* matrices, r is the *rank* of matrix S, and $\sum$ is a *diagonal* matrix of size $r \times r$ of which its diagonal entries contain all singular values of matrix S that are sorted in decreasing order.

SVD provides the best lower rank approximation of the original matrix S [2]. The diagonal matrix $\sum$ can be reduced into a lower-rank diagonal matrix $\sum_{k\times k}$ by retaining the k $(< r)$ largest values. We reduce U to $U^{'}$ and V to $V^{'}$. Sequentially, the original matrix S is reduced to

$$S^{'} = U^{'} \bullet \sum\nolimits^{'} \bullet \ V^{'} \tag{6}$$

which is the rank-k approximation of S.

In the reduction process, $U^{'}$ consists of the first k columns of the matrix U, which corresponds to the k highest order singular values. In the resulting semantic attribute matrix, $S^{'}$, each item is represented by a set of k latent attributes (i.e., terms). The results yield a much less sparse matrix, which improves the outcomes of similarity computations and the computational cost throughout the process. Moreover, the generated latent attributes represent groups of highly correlated attributes in the original data which potentially reduce the amount of noise associated with the semantic information. Performing latent semantic analysis on the semantic space can generally lead to substantial gains in prediction accuracy based on the semantic attributes, which we have applied to our long-tail recommendation strategy to (i) minimize the number of latent information to be considered, and (ii) speed up the recommendation process in terms of processing time.

IV. Experimental Results

To analyze the performance of our extended tripartite algorithm, the recall and diversity tests were run on the same MovieLens dataset used by Johnson et al.'s algorithm [7]. We compare the scores achieved by our extended tripartite graph algorithm with Johnson et al.'s original tripartite algorithm, in addition to Yin's bipartite graph algorithms [15] and Shang et al. [13] tripartite graph-based algorithm.

A. Existing Bipartite and Tripartite Algorithms

Prior to presenting any experimental results, we present the formal approach of Markov process used by Johnson et al [7]. Hereafter, we formally introduce Yin's bipartite graph, and Shang et al.'s tripartite graph algorithm for long-tail item recommendations for comparison purpose.

1) Markov Chains: Critical to the long-tail solution proposed by Johnson et al. [7] is the balance of a trade-off between the implementations of the tripartite graph and the Markov chain or Markov process. A *Markov process* describes the *probabilities* of a random walker arriving at a node j from a node i, which is termed as $p_{i,j}$, given an increasing time horizon T. As T increases, the elements of a stochastic transition matrix L^2 converge to probabilities that do not change with subsequent iterations of the *Markov process*. This state is known as the *stationary distribution*.

Specifically, while the tripartite graph adds a significant number of *shorter paths* between user u and a preferred item i and the Markov process calculates the probabilities of u arriving at items that (s)he has not yet rated, the process will favor items with more links as the time T for the Markov chain increases. In other words, as T increases so does the probability of arriving at an item with few ratings—a long-tail item. Johnson et al. provide specific details of the implementation of the tripartite graph and Markov chain and describe in more detail the benefit they provide the solution (see implementation details in [7]).

The experimental results, as shown in [7], illustrates that Johnson et al.'s recommendation algorithm works best for the lower values of T. As the Markov process progresses towards the stationary probability, the descriptiveness of the graph is lost. For T = 7, the ability of the model to recommend long-tail items breaks down. Johnson et al. demonstrate in [7] that based on their results and those archived by the absorbing cost (AC) methods in [15], their proposed tripartite graph recommendation system edges out the AC methods at both T = 3 and T = 5.[3]

2) Absorbing Cost and Hitting Time: A score rated by a user who specializes in limited interests often provide much more valuable information than the rating offered by a person of wide interests. The bipartite-graph based Absorbing Cost (AC) algorithms proposed by Yin et al. [15] compute such a score as defined below.

$$AC(S|i) = \begin{cases} 0 & i \in S \\ \sum_j p_{ij} \times c(j|i) + \sum_j p_{ij} \times AC(S|j) & i \notin S \end{cases} \quad (7)$$

where p_{ij} is the *probability* of following a path from node i to node j, $c(j|i)$ is the *transition cost* from state i to its adjacent state j, and S, called *absorbing nodes*, denotes the set of items rated (or purchased) by a query user. AC is the average cost incurred by the random walker starting from state i to reach S for the first time. While the hitting time (HT) algorithm (introduced in Section II) employes the ratings that users gave to items, AC entails other features.

The HT algorithm [15] considers the number of steps a user i takes to reach a product j in a graph. The single-step transition *probability* from i (at time t) to j (at time $t+1$) is

$$p_{i,j} = P(s(t+1) = j \mid s(t) = i) = \frac{a(i,j)}{d_i} \quad (8)$$

where d_i=$\sum_{k=1}^{n} a(i,k)$, and $a(i,k)$ is the weight of i and k.

While traversing a graph for query user q, HT is given by

$$H(q|j) = \frac{1}{p_{j,q}} = \frac{\pi_j}{p_{q,j}\pi_q}, \text{ and } \pi_i = \frac{\sum_{j=1}^{n} a(i,j)}{\sum_{i,j=1}^{n} a(i,j)} \quad (9)$$

The metric $H(q|j)$ has the unusual quality that a *low* hitting time means q and j are *relevant* and *few* users have rated item j. In other words, j is a product that is similar to those q has rated high and j is in the long tail. This ingenious approach maps q to items that are in the long tail and potentially appealing to q.

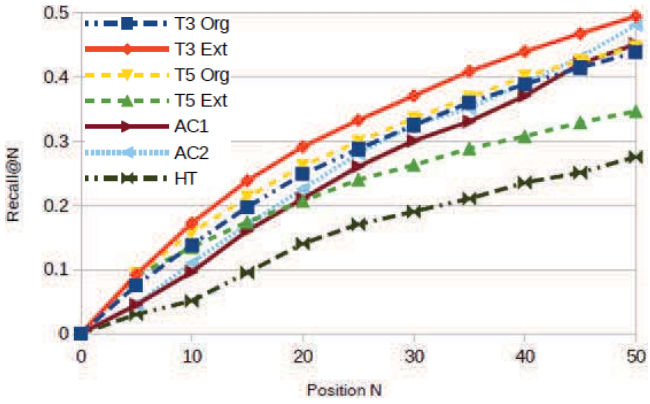

Figure 4. *Recall@N* values of various versions of bipartite and tripartite graph structure

B. Recall/Diversity Values of Various Algorithms on Long-Tail Recommendations

The results achieved by using the extended tripartite approach on long-tail recommendations are shown in Figure 4 and Table I. Overall, the best performing algorithm is the

[2]A stochastic transition matrix indicates the probabilities of different nodes connected in an undirected graph based on edges connected among different nodes in the graph.

[3]$T3, T5$, and $T7$ referring to the Markov process where T = 3, 5, and 7, respectively.

Table I
Diversity VALUES OF VARIOUS VERSIONS OF BIPARTITE AND TRIPARTITE GRAPH STRUCTURE

Algorithm	Diversity Score
T3 Original [7]	0.404
T5 Original [7]	0.315
T3 Extended (Ours)	**0.450**
T5 Extended (Ours)	0.325
AC1 [15]	0.425
AC2 [15]	0.420
HT [15]	0.410

$T3$ Extended algorithm, i.e., when T = 3, which improves on the previous best tripartite graph-based algorithm, i.e., $T5$ Original [7], on the $Recall@N$ score by around 9.5%, which is statistically significant ($p < 0.01$) [9]. Figure 4 also shows the results of Yin's *Hitting Time* algorithm, HT, and *absorbing cost* algorithms, $AC1$ and $AC2$, which are outperformed by the $T3$ Extended algorithm.

The *diversity* score for the extended tripartite algorithm and the scores from other algorithms are shown in Table I. The $T3$ Extended algorithm beats the previous best score of AC1 by 2.5%, which is statistically significant ($p < 0.05$).

C. The Rankshift Test on Long-Tail Items

Recall that the long-tail problem describes the difficulty of making appropriate recommends to users and for items we have little data about. To further test the effectiveness of the extended tripartite graph recommender on long-tail items, we design a new *rankshift test*, which measures how the relative recommendation strength of an item changes as we change it from a short-head item to a long-tail item. The test generates recommendations using the extended tripartite graph algorithm and measures where in ranking a short-head item falls. Rating data is then removed from the initial data set to make the short-head item a long-tail item. Recommendations are then generated again, and the rankings of the newly-created long-tail items are compared with the original rankings, which yields a measure of how much weighting the algorithm gives to less-popular items, and provides insight into how the system treats long-tail items. We have seen based on the recall and diversity tests that the extended tripartite graph system is able to recommend long-tail items, which are items with little to no data associated with them, again relying on latent information to fill in gaps in the data.

In order to better understand how the extended tripartite graph algorithm handles recommendations of long-tail items, we provide the *rankshift algorithm* (given below) which measures how the recommendation strength for an item changes when most of its rating data is removed.

1. Select an item to perform the rankshift test on.
2. Determine all users who have not rated this item, and designate them as test users.
3. Generate item recommendations for all test users.
4. Alter rating information so that only one rating remains for the item being tested.
5. Generate item recommendations for all test users, using the altered rating data.
6. For each user, compare where the item falls in recommendation ranking using the original data versus the altered rating data.
7. Take an average of these comparison values.

Table II
SHIFT IN RANK FOR VARIOUS ITEMS

Item ID	Original Percentile	New Percentile	Change
534	40.41	95.73	55.32%
1000	61.84	98.11	36.27%
1196	30.28	99.02	68.74%
2236	38.76	92.85	54.09%
3068	41.34	97.20	55.86%

The comparison values from Step 7 provide insight into whether the extended tripartite algorithm treats the low-rated data items differently than their regular counterparts. If ranking does not change significantly between the two, it could be concluded that whether an item is a long-tail item or not does not affect how strongly it is recommended.

The data in Table II, which was collected by running the extended tripartite algorithm at T = 3, shows the change in rank by percentiles, where the "New Percentile" column refers to the average ranking of the item when it is a long-tail item. The *higher* the percentile, the *stronger* an item is recommended to a user, i.e., an item recommended at the 100^{th} percentile is the item the algorithm recommends most strongly to a user.

Table II shows that the extended tripartite graph recommender system very heavily favors recommending items with less ratings. Regardless of what percentile the item started at, changing it to be a long-tail item gives it a huge boost in rankings—the item very often ends up in the top 5^{th} percentile of recommendations. One of the design goals of the extended tripartite graph recommender is to suggest long-tail items to users, as explained in Section I, and just recommend the same popular items to all users is not desirable. The results show that the algorithm is indeed favoring long-tail items over items which are more popular.

In order to have a better idea of how the ranking of an item changes as the number of ratings decreases, we tweak the original rankshift algorithm. Instead of keeping one rating, we keep a subset of ratings. We also skip Steps 6 and 7, which compares how much the percentile changed, and instead look at the current percentile. Let's consider data item 1196 for the subset of ranges between 150 and 1 rating(s) as shown in Figure 5.

As seen in Figure 5, the item percentile gradually in-

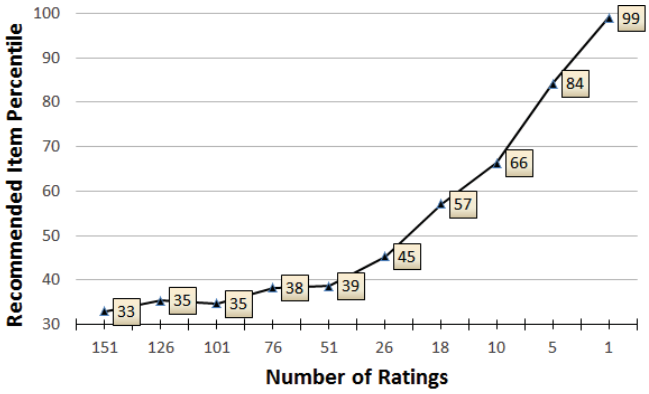

Figure 5. Ratings versus Item Ranking

creases as the number of ratings decreases. Once the number of ratings falls below 25 or so, the change in percentile accelerates climbing. The item approaches the 100^{th} percentile as the number of ratings approaches 1. This further confirms that the extended tripartite graph algorithm prefers recommending long-tail items over items with more ratings.

D. The Recall, Diversity, and Rankshift Tests

Consider what the results of the recall, diversity, and rankshift tests impose on the extended tripartite algorithm. The *recall* test shows that the algorithm is correctly learning the *preferences* of users, and is recommending items that fit that preference. The *diversity* test shows that the extended tripartite algorithm is not just recommending the same items over and over again. Instead, it recommends a wide set of items across users. This implies that the algorithm is not just looking at what items are highly rated, but individualizes recommendations based on the user it is generating recommendations for. Finally, the *rankshift* test shows that the algorithm has a strong preference for long-tail items over more popular items. The results of the three tests validate that the extended tripartite algorithm learns the preferences of the users and makes individualized recommendations.

V. Conclusions

The works of Yin et al. [15] and Shang et al. [13] have shown that a graph-based approach is a potential solution to the long-tail problem. The tripartite recommendation algorithm proposed by Johnson et al. [7] have improved upon the results of Yin's, and our extended tripartite recommender have further enhanced the performance of Johnson et al.'s recommendation approach. Our improvements come from carefully considering the original tripartite algorithm introduced by Johnson et al. and treating the data for finding long-tail items. We introduce the concept of "basic genres" to create more paths in the tripartite graph, and we switch from using a simple average to a Bayesian average to avoid giving too much weight to lesser-rated genres. By reworking the original algorithm proposed by Johnson et al. to better utilize the user and item data, we improve the recall and diversity scores beyond the work already done.

Furthermore, we have also proposed to adopt Singular Value Decomposition used in Latent Semantic Indexing to solve the long-tail recommendation problem. The proposed solution to the long-tail recommendation is elegant, since it relies on the notable technique in matrix decomposition, using the latent semantic analysis approach to utilize the latent information to fill in gaps in the user-item data.

The goal of our recommender is to make good long-tail suggestions. Item should be suggested not because it is a long-tail item, but because it is a long-tail item that matches a user's preferences.

References

[1] G. Adomavicius and A. Tuzhilin. Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions. *IEEE TKDE*, 17(6):734–749, June 2005.

[2] M. Berry, S. Dumais, and G. Brien. Using Linear Algebra for Intelligent Information Retrieval. *SIAM Review*, 37:573–595, 1995.

[3] C. Clarke, M. Kolla, G. Cormack, O. Vechtomova, A. Ashkan, S. Buttcher, and I. MacKinnon. Novelty and Diversity in Information Retrieval Evaluation. In *ACM SIGIR*, pages 659–666, 2008.

[4] A. Eberle. Markov Processes. University of Bonn, Institute for Applied Mathematics, 2015.

[5] J. Herlocker, J. Konstan, and J. Riedl. An Empirical Analysis of Design Choices in Neighborhood-based Collaborative Filtering Algorithms. *Information Retrieval*, 5:287–310, October 2002.

[6] T. Hofmann. Probabilistic Latent Semantic Indexing. *ACM SIGIR Forum*, 51(2):211–218, July 2017.

[7] J. Johnson and Y.-K. Ng. Enhancing Long Tail Item Recommendations Using Tripartite Graphs and Markov Process. In *IEEE/WIC/ACM WI'17*, pages 761–768, 2017.

[8] V. Klema and A. Laub. The Singular Value Decomposition: Its Computation and Some Applications. *IEEE Transactions on Automatic Control*, 25(2):164–176, April 1980.

[9] C. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge, 2008.

[10] M. McLaughlin and J. Herlocker. A Collaborative Filtering Algorithm and Evaluation Metric that Accurately Model the User Experience. In *ACM SIGIR*, pages 329–336, 2004.

[11] Y.-J. Park and A. Tuzhilin. The Long Tail of Recommender Systems and How to Leverage It. In *ACM RecSys*, pages 11–18, 2008.

[12] F. Ricci, L. Rokach, B. Shapira, and P. Kantor. *Recommender Systems Handbook*. Springer, 2011.

[13] M. Shang, Z. Zhang, T. Zhou, and Y. Zhang. Collaborative Filtering with Diffusion-based Similarity on Tripartite Graphs. *Physica A: Statistical Mechanics and Its Applications*, 389(6):1259–1264, 2010.

[14] X. Yang and Z. Zhang. Combining Prestige and Relevance Ranking for Personalized Recommendation. In *ACM CIKM*, pages 1877–1880, 2013.

[15] H. Yin, B. Cui, J. Li, J. Yao, and C. Chen. Challenging the Long Tail Recommendation. In *VLDB Endowment Vol. 5, No. 9*, pages 896–907, 2012.