Random Walks in Recommender Systems: Exact Computation and Simulations*

Colin Cooper Sang Hyuk Lee Tomasz Radzik

Yiannis Siantos

Department Of Informatics
King's College London, U.K.

ABSTRACT

A recommender system uses information about known associations between *users* and *items* to compute for a given user an ordered recommendation list of items which this user might be interested in acquiring. We consider ordering rules based on various parameters of random walks on the graph representing associations between users and items. We experimentally compare the quality of recommendations and the required computational resources of two approaches: (i) calculate the exact values of the relevant random walk parameters using matrix algebra; (ii) estimate these values by simulating random walks. In our experiments we include methods proposed by Fouss *et al.* [7, 8] and Gori and Pucci [10], method P^3, which is based on the distribution of the random walk after three steps, and method P^3_α, which generalises P^3. We show that the simple method P^3 can outperform previous methods and method P^3_α can offer further improvements. We show that the time- and memory-efficiency of direct simulation of random walks allows application of these methods to large datasets. We use in our experiments the three MovieLens datasets.

1. INTRODUCTION

We view a recommender system as an algorithm which takes a dataset of relationships between a set of *users* and a set of *items* and attempts to calculate how a given user might rank all items. For example, the users may be the customers who have bought books from some (online) bookstore and the items the books offered. The core information in the dataset in this case would show who bought which books, but it may also include further details of transactions (the date of transaction, the books bought together, etc.), information about the books (authors, category, etc.), and possibly some details about customers (age, address, etc.). For a given customer, a recommender system would compute a list of books $\langle m_1, m_2, \ldots, m_k \rangle$ which this customer might be interested in buying, giving the highest recommendations first. Recommender systems viewed as algorithms for computing such personalised rankings of items (rather than overall "systems," which would also include methods for gathering data) are often referred to as *scoring*, or *ranking*, or *recommender algorithms*.

In this paper, we focus on a simple scenario with two main entity sets, Users (U) and Items (I), and a single relationship R of pairs $\langle u, m \rangle$, where $u \in U$ and $m \in I$. The fact that $\langle u, m \rangle \in R$ means that u has some preference for m. The pairs $\langle u, m \rangle \in R$ may have additional attributes which indicate the degree of preference. The relationship R can be modeled as a bipartite graph $G = (U \cup I, R)$, possibly with edge weights, which would be calculated on the basis of the attributes of pairs $\langle u, m \rangle \in R$. A scoring algorithm for a user $u \in U$ orders the items in I according to some similarities between vertex u and the vertices in I, which are defined by the structure of graph G. More precisely, a scoring algorithm is defined by a formula or a procedure for calculating a $p \times q$ matrix $M = M(G)$ which expresses those similarities, where $p = |U|$ and $q = |I|$. For a user $u \in U$, the items $m \in I$ are ranked according to their $M(u, m)$ values (in increasing or decreasing order, depending whether a lower or a higher value $M(u, m)$ indicates higher or lower similarity between vertices u and m). Matrix M is called the *scoring matrix* or the *ranking matrix* of the algorithm.

Fouss *et al.* [7, 8] proposed various scoring algorithms which are based on parameters of random walks in G, or more generally on the Laplacian matrix of G and compared them with some other methods based on matrix operations. They evaluated their performance on a dataset of movie ratings gathered by the MovieLens website [4]. There are three MovieLens datasets, containing 100K, 1M, 10M entries, respectively. Each entry is a quadruple $\langle$*UserId, MovieId, Rating, Timestamp*$\rangle$, which we view as pair $\langle$*UserId, MovieId*$\rangle$ with attributes *Rating* and *Timestamp*.

Let $\mathcal{W}_u = \langle W_u(0), W_u(1), \ldots, W_u(t), \ldots \rangle$ be a random walk on graph G starting from vertex u. That is, $W_u(0) = u$ and $W_u(t+1)$ is a randomly selected neighbour of $W_u(t)$. Let $h(u, v)$ denote the first step $t \geq 1$ such that $W_u(t) = v$.

*This research is part of the project "Fast Low Cost Methods to Learn Structure of Large Networks," supported by the 2012 SAMSUNG Global Research Outreach (GRO) program.

WWW'14 Companion, April 7–11, 2014, Seoul, Korea.
ACM 978-1-4503-2745-9/14/04.
http://dx.doi.org/10.1145/2567948.2579244.

The *hitting time* of v from u is the expectation $H(u,v)$ of the random variable $h(u,v)$. The *Laplacian matrix* of graph G is defined as $L = D - A$, where A is the adjacency matrix of G and D is the diagonal matrix of the vertex degrees. The main scoring algorithms considered in Fouss *et al.* [7, 8] are the hitting-time, the reverse hitting-time, the commute-time and the L-pseudoinverse algorithms. The scoring matrices M of these algorithms are obtained from matrices H, H^T, $C = H + H^T$ and L^+, respectively, where L^+ is the *Moore-Penrose pseudoinverse* of L, (also referred to as the *inverse Laplacian*). Here, a lower value of $M(u,m)$ indicates higher similarity between vertices u and m, and a higher rank of m in the ranking list of items for the user u.

Fouss *et al.* [7, 8] reported that for the 100K MovieLens dataset which they used in their experiments, the L^+ algorithm (and its variants) performed the best. The hitting-time and the commute-time algorithms performed similarly to the simple user-independent algorithm which orders the items according to their degrees in G (that is, ordering according to their popularity). The reverse hitting-time algorithm was the worst.

Our work expands Fouss *et al.* [7, 8] in several ways. We consider simpler and faster scoring algorithms and show that they can match, and sometimes exceed, the best performance of the previous algorithms. Our experiments show a good performance of the simple and intuitive method P^3, which ranks the items for a user u according to the distribution of the third vertex on the random walk starting at vertex u. Equivalently, P^3 stands for the third power of the transition matrix $P = D^{-1}A$ of a random walk. An item m_1 gets a higher rank than an item m_2, if $\mathbf{Pr}(W_u(3) = m_1) \geq \mathbf{Pr}(W_u(3) = m_2)$. We generalise method P^3 to a parameterised method P^3_α and demonstrate that P^3_α, for an empirically optimised value α, can offer further improvements. Method P^3_α is based on the third power of the matrix P_α with entries equal to the entries in matrix P raised to the power of α.

While including in our experiments the 100K MovieLens dataset which was used in [7, 8], to be able to compare results, we also use the medium and the large MovieLens datasets with 1M and 10M entries (not used in [7, 8]) to see how the performance of ranking algorithms scales up. We also expanded the evaluation of performance of ranking methods by adding a measure of result quality, based on number of correct entries among the top recommendations.

Our experimental results show, for example, that methods L^+ and P^3 perfom equally well (and clearly better than the other methods) on the small 100K dataset, but P^3 outperforms L^+ on the medium 1M dataset. For the large 10M dataset, method P^3 again shows the best performance, while method L^+ could not be tested because the memory and computational time requirements were too high. We do not include here details, but we note that the memory needed to obtain the matrix P^3 for the 10M MovieLens dataset via matrix operations was $64GB$ while $\hat{P}^3$ was obtained by simulating random walks and required approximately $2GB$.

The exact values of the relevant parameters of random walks can be obtained by algebraic matrix manipulations, but this is costly in time and memory. Therefore we estimate these parameters by simulating short random walks. In common with our findings that P^3 gives best results among matrix based methods, we find that random walks of length 3 give best results. We also compare the performance of the P^s method, $s = 3, 5$, which computes the s-th power of matrix P, with the performance of the $\hat{P}^s$ method, which estimates the s-th power of P by simulating random walks of length s. We observe, for example, that in the small dataset $\hat{P}^3$ approximates P^3 with a relative difference 0.05 after $20n$ walk steps (that is, $6.7n$ random walks of length 3, where n is the number of vertices in G). The large dataset reaches this accuracy in less than n steps.

Recommender algorithms based on the third neighbourhoods of vertices in G, including methods P^3 and #3-Paths (see Sec. 4.1), are actually widely used in practice. It is still not clear, however, what is the best way of using the structure of those neighbourhoods. Our experimental results indicate that the simple method P^3_α can offer further clear improvements over the P^3 and #-3-Paths methods.

2. RELATED WORK

Kunegis and Schmidt [12] extend Fouss' work of [8] by taking user ratings into account while computing a similarity measure on *users-items* bipartite graph. The similarity measure used in the paper is resistance distance, which is equivalent to the commute-time distance used in [8]. They adapt a rescaling method proposed in [5], in which the user ratings are rescaled into 1 (good) to -1 (bad). The authors conduct experiments on two datasets: 100K MovieLens dataset [4] and Jester [1] and the performance is measured by two evaluation metrics, the *mean squared error* and the *root mean squared error*.

Gori and Pucci [9, 10] present a random walk based scoring algorithm. In [10] the algorithm is applied to movie recommendation whilst in [9], the algorithm is used for recommending research papers. For the movie recommendation, 100K (small) MovieLens dataset is used for the experiments and for the research paper recommendation, they use the dataset that is derived from the crawling of ACM portal website. The algorithm is based on the Pagerank algorithm applied to an item similarity graph called a correlation graph. The correlation graph is constructed in [10] from the *users-items* bipartite graph, and in [9] from *citation* graph. Zhang *et al.* [20, 19] extend Gori and Pucci's work [10] by taking into account user's preference on item categories. The proposed algorithm computes the ranking scores based on the item genre and user interest profile.

Craswell and Szummer [6] consider a recommender algorithm based on the distribution P^j_δ of the j-th step of the random walk with the probabilities of self-transitions equal to δ. In their experiments on click graphs, the best performance was achieved when $j \approx 100$ and $\delta = 0.9$. For $\alpha > 1$, the method P^3_α which we consider in this paper can be defined in terms of random walks with positive self-transition probabilities, which are different for different vertices. Thus, P^3_α is different from P^3_δ as the self-transition probabilities in the latter are the same for all vertices.

Singh *et al.* [18] present an approach of combining a relations graph (friendship graph) with the ownership data (user-item graphs) to make recommendations. The combined graph is represented as an augmented bipartite graph and is treated as a Markov chain with an absorbing state. The items are ranked according to the approximated absorbing distribution. This method is tested and evaluated for on-line gaming recommendation.

Lee *et al.* [13] consider a multidimensional recommendation problem, which allows some additional contextual in-

formation as an input. They present a random-walk based method, which adapts the Personalized PageRank algorithm. They conduct experiments on two datasets: *last.fm* [2] and *LG's OZ Store* [3]. The performance is evaluated using *hit at top-k* evaluation metric. Further work in this direction is reported in [14, 15].

3. EVALUATION METHODOLOGY

To ensure reproducibility of results and to facilitate comparison between various recommender systems, each MovieLens dataset is partitioned into two parts: a training, or base, set B and a test set T. The test set is obtained by selecting 10 random user-movie ratings for each user. The training set consists of all remaining ratings. A recommender system is run on the training set to compute for each user a ranking of movies, and then the computed rankings are compared with the test set. A better recommender system would be more effective in reconstructing the hidden information, that is, would give a closer fit between the computed rankings and the test set. It is not obvious, however, how the closeness between the computed rankings and the test set should be quantified and a number of measures have been proposed (See, for example, Herlocker *et al.* [11]).

Among the most common ones are the percentage of *correctly placed pairs* (used in [7, 8]) and the number of *hits in the top k recommendations*. We use both these measures in this paper, referring to them as *Metric I* and *Metric II*, respectively. For a ranking algorithm $\mathcal{A}$ and a user u, a pair of items $\langle m', m'' \rangle$, where m' is in this user's test set and m'' in the training set, is "correctly placed" in the ranking list computed by $\mathcal{A}$ for user u, if m' is higher than m''. For the top k recommendations measure (Metric II), we use relative values for k (as fraction of the number of items) since this is more appropriate when using datasets of varied sizes.

We experimented with all three MovieLens datasets, but because of the space limit, in most cases we present only the results for the small and large datasets. We have also repeated our experiments for various randomly selected test sets T observing each time very similar evaluation scores which always gave the same order of the methods. We therefore use the same B and T used in [7, 8].

4. MATRIX BASED METHODS

4.1 Ranking algorithms

The input for the ranking methods is the bipartite graph $G = (U \cup I, R)$. The methods are general, but we describe them using the terminology of the MovieLens datasets, since our evaluation are based on those datasets. Thus we refer to U as the set of users, to I as the set of movies, and the edges in G show which movies the users have watched. A ranking algorithm computes for each user a ranking of movies.

We have experimented with a number of ranking methods, looking for simple and intuitive methods which would match, or ideally exceed, the performance of the methods investigated in Fouss *et al.* [7, 8]. We include in this paper experimental results for five ranking methods (Ranking by degree (MaxF); Hitting-time ($Hit^{\rightarrow}$); Reverse hitting-time ($Hit^{\leftarrow}$); Average Commute time (AVC); Pseudo-Inverse Laplacian (L^+)) considered in [7, 8], the ItemRank (IR) method described in [10], and the following methods.

The s-step random walk distribution ($\mathbf{P^s}$). Movies that the user u has not watched are ranked based on the probability distribution $P^s(u,.)$ of the random walk at step s, if u was the starting vertex. If $P^s(u,m') > P^s(u,m'')$, then movie m' is ranked higher than movie m''. Matrix P^s is the s-th power of the transition probabilities matrix of the random walk on G. Observe that only odd numbers $s \geq 3$ should be considered: G is bipartite so for a user u and a movie m not watched by u, the probability $P^s(u,m)$ can be positive only for an odd $s \geq 3$. In this paper we include experimental results only for the P^3 and P^5 rankings.

Number of paths of length 3 (#3-Paths). Movies that the user has not watched are ranked based on the number of distinct paths of length 3 from that user to the movies in graph G. A movie m with the greater number of paths has a higher ranking.

Inverse Laplacian of the transition matrix ($\mathbf{Z/\pi}$). Matrix Z is defined by letting

$$Z_{ij} = \sum_{t \geq 0} \left(P^{(t)}(i,j) - \pi_j \right), \tag{1}$$

where $P^{(t)}(i,j)$ is the probability that the random walk starting at vertex i is at vertex j at step t, and π is the stationary distribution. For a user u, the Z/π method ranks the movies according to their $Z_{u,m}/\pi_m$ values, with higher values indicating higher ranks. Matrix Z appears in some identities characterizing the parameters of the random walk. For example, Z_{ij}/π_j is the expected hitting time of j from the stationary distribution minus the expected hitting time of j from i. Matrix Z can be obtained from the inverse Laplacian of the transition matrix P: $Z = (I - P + \Pi)^{-1} - \Pi$, where I is the identity matrix and $\Pi_{ij} = \pi_j$ [16].

4.2 Experimental results for matrix methods

In this section we present our experimental results for the matrix based ranking methods. Tables 1 and 2 show the performance of the ranking algorithms on the small and large MovieLens datasets evaluated with Metric I and Metric II. In Table 1 we obtained the same metric I scores as in [7, 8] for the methods considered there. Regarding the additional methods, the P^3 ranking method turns out to perform very well, matching, and improving on the performance of the L^+ method. We note that for the large dataset, due to insufficient memory space the ranking algorithms, $\text{Hit}^{\leftarrow}$, $\text{Hit}^{\leftarrow}$, AVC, L^+, IR, and Z/π are not included in the experiments.

Tables 1 and 2 also show the scores of the ranking algorithms obtained using the metric II. We varied the parameter k of this metric from 1% to 10%. It turns out that the relative performance of the ranking algorithms in our experiments is exactly the same in both metrics. The scores computed by the metric II, however, seem to be easier to interpret. For example, the metric II scores in Table 1 clearly show that the reverse hitting time ranking is much weaker than the hitting time ranking.

4.3 Parametrised method $\mathbf{P^3_\alpha}$

In this section we present a method for improving the performance of P^s by introducing a parameter α, which ranges over the real numbers. We define matrix P_α as derived from the transition probability matrix P by raising every positive entry to the power of α.

We rank the movies for the users using matrices P^3_α with the values of α ranging from 0 to 4 in steps of 0.1. The

Small size Dataset										
Evaluation	**MaxF**	**Hit$^{\rightarrow}$**	**Hit$^{\leftarrow}$**	**AVC**	**L^+**	**IR^1**	**Z/π**	**P^3**	**P^5**	**3-Paths**
Metric I	85.79%	85.74%	80.45%	85.76%	90.94%	89.03%	85.72%	**90.99%**	88.19%	87.95%
MetricII@1%	0.169	0.161	0.007	0.161	0.214	0.212	0.160	**0.261**	0.197	0.193
MetricII@3%	0.294	0.293	0.051	0.294	0.445	0.376	0.292	**0.452**	0.362	0.350
MetricII@5%	0.381	0.382	0.125	0.382	**0.582**	0.478	0.381	0.559	0.460	0.444
MetricII@10%	0.543	0.543	0.341	0.544	**0.736**	0.637	0.542	0.699	0.616	0.609

Table 1: Small-size dataset (User, Movie).

Large size Dataset				
Evaluation	**MaxF**	**P^3**	**P^5**	**#3-Paths**
Metric I	93.94	**95.98**	94.64	94.69
MetricII@1%	0.353	**0.481**	0.405	0.407
MetricII@3%	0.569	**0.680**	0.604	0.607
MetricII@5%	0.679	**0.778**	0.714	0.714
MetricII@10%	0.816	**0.885**	0.839	0.842

Table 2: Large size dataset (User, Movie).

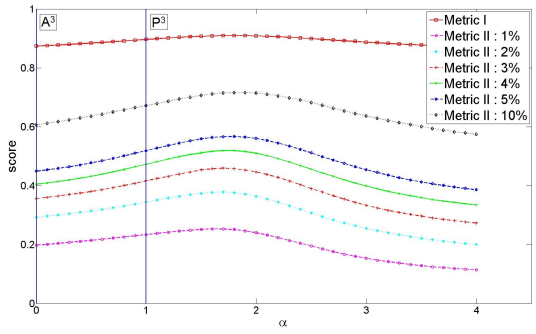

Figure 1: Medium Dataset: Optimisation of the performance of P^3 using parameter α

quality of the obtained rankings for the small MovieLens datasets is plotted in Figures 1. The quality is measured in both Metric I and II. Note that the vertical lines at $\alpha = 0$ and $\alpha = 1$ denote the performance of the #3-Path and P^3 ranking algorithms, since $P_0 = A$ and $P_1 = P$, respectively. For the medium dataset, the performance measured in Metric I is improved from 89.62% to 90.90% for $\alpha = 1.9$ The performance evaluated in Metric II is also increased compared to the performance of P^3.

Note that of $\alpha \neq 1$, then P_α is not a transition probability matrix. It is possible to estimate entries of P^3_α using short random walks, but we do not consider in this paper.

The P^3 and P^3_α methods compute recommendations for a user u based only on the neighbourhood of u in graph G of diameter 3. These methods perform well probably because the diameter of G is small, e.g. the small dataset has diameter 6.

5. RANKING METHODS BASED ON SIMULATION OF RANDOM WALKS

Ranking algorithms based on matrix manipulations have their natural limit: the sizes of the matrices may quickly become too large to fit in the computer memory. We did not have problems with running the ranking algorithms on the small MovieLens dataset, but some technical issues of insufficient memory became a major obstacle for computing matrices in the largest MovieLens dataset.

An alternative approach is to gather information about the structure of a graph from simulations of random walks in this graph. For example, the hitting times used in the ranking methods in the previous section can obviously be estimated by simulating random walks. Such simulations require only $O(r)$ space, to store the adjacency lists of the graph, with $r \ll n^2$ for sparse graphs (n is the number of vertices and r is the number of edges). Furthermore, the computation can be organised in a distributed manner, if the graph is scattered over a network.

5.1 Data capture

We consider ranking methods which compute the ranking of items for a user u on the basis of the information gathered by random walks starting from vertex u. We set the limit of n on the total number of steps of all random walks performed for one user (called the "budget"). More precisely, we perform $w = n/s$ random walks starting from a given vertex u, with each walk having s steps, for some fixed parameter s. Taking into account all w walks, we rank vertices on the basis of how often they are hit on average, or how quickly they are hit on average. For each vertex v and each random walk $i = 1, 2, \ldots, w$, we keep the following information: (a) the step $s_v(i)$ when vertex v was visited by walk i for the first time. (b) the indicator $n_v(i)$, which is equal to 1, if vertex v was visited by walk i, and 0 otherwise, and (c) the degree $d(v)$ of vertex v.

5.2 Random Walks: Evaluation Methodology

Similarly to the work of Sarkar and Moore [17], we estimate the *truncated hitting time* $h^T(u, v)$ of vertex v starting from vertex u by $\frac{1}{w}\sum_{i=1}^{w} s_v(i)$. To use this estimator, we have to decide what should be the value of $s_v(i)$, if vertex v was not visited during walk i. Vertices which have not been visited by any of the walks are not ranked so we do not need to worry about their $s_v(i)$ values. We consider now a vertex v which was visited by some walks, but was not visited by, say, walk i. Some hitting time penalty should be imposed for unvisited vertices, and we experimented with various values for this penalty. (a) Set the penalty to $\hat{r}$, our estimate of

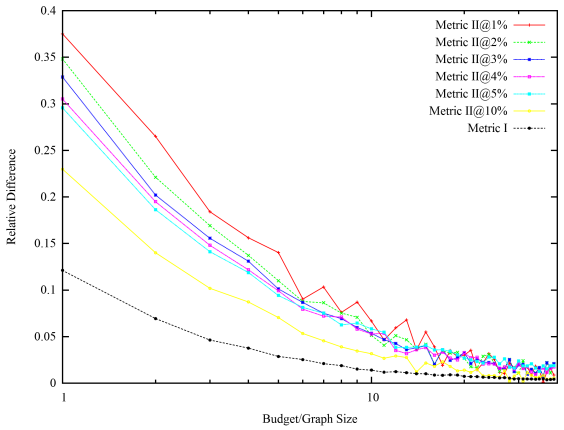

Figure 2: Convergence on small dataset

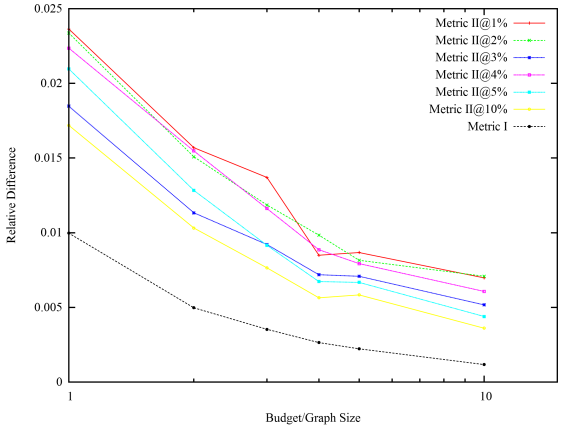

Figure 3: Convergence on large dataset

the number of edges incident with the subgraph we can visit in s steps. (b) Set the penalty to $2\hat{m}/d(v)$ the approximate the *first hitting time* of a vertex. We do not use the total number of edges of a graph to cover the cases when this is unknown. (c) Set the penalty to s, the walk length. This penalty used in [17]. (d) Set the penalty to zero.

Having estimated $h^T(u,v)$ and using one of the above penalties, we can rank the movies in ascending order of their $h^T(u,v)$ values. In addition to estimates of $h^T(u,v)$ we rank movies based on the number of times they were hit on average. We calculate the average number of hits of a vertex v from vertex u as $n(u,v) = \frac{1}{w}\sum_{i=1}^{w} n_v(i)$. For example, if a movie m is hit $w/2$ times by w walks, then the number of hits is 0.5. Intuitively, movies which were hit often are more relevant to a user and should be ranked higher. Movies are ranked for recommendation in the descending order according to their average number of hits. We also consider some reweighting heuristics, such as (number of hits)*(degree).

We also rank movies based on an estimate of the s step transition probability. We can estimate this simply by counting $\hat{P}^{(k)}(u,v)$ – the number of walks which hit vertex v at step s divided by the total number of walks w. This value converges to P^s considered in Section 4 if a large number of walks are run.

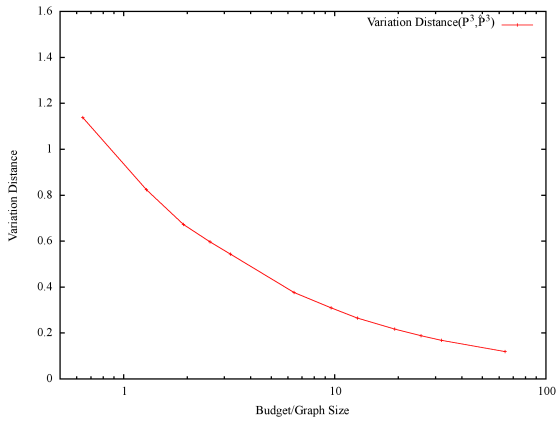

Figure 4: Total variation distance

5.3 Random walks: Experimental results

In order to estimate random walk properties without using matrix operations we made use of many short random walks of various lengths. There were W random walks performed, each of length s. Each individual walk visited a number of vertices within s steps of the starting user. For each of these vertices, if they corresponded to movies, we recorded the number of times they were visited by the walk at a specific step. All movies were evaluated based on properties such as Truncated hitting times, number of Hits, number of hits at step s then ranked accordingly. The ranking obtained through each method was evaluated using both metrics described in Section 3. For most experiments, $s = 3, 5, 7, 9$. The values of W varied according to s based on the budget/step allowance of each experiment. For the results in Tables 3-4 the budget was set to be the number of vertices in the graph n. Walks of length $s = 7, 9$ had diminishing scores and due to space limitations are not included.

The method $\hat{P}^3$ has the best performance. The method based on the number of hits weighted with the vertex degrees comes second. Among the methods based on the truncated hitting times, the penalties $\hat{r}$ and $2\hat{r}$/degree, combined with s equal to 3 or 5, led to reasonably good performance. The scores of these two methods are very close to the scores of the hitting time method Hit$^{\rightarrow}$ given in Table 1.

5.4 Convergence of 3-step random walks to P^3

We experimented with method $\hat{P}^3$ for varying budget $b = cn$ of the short random walks. We compared results with the matrix multiplication method P^3 (as seen in Table 1). The comparison is done using the formula:

$$\text{Relative difference} = \frac{|\text{Score of } \hat{P}^3 - \text{Score of } P^3|}{\text{Score of } P^3} \qquad (2)$$

The results for the small dataset can be seen in Figure 2, and for the large dataset in Figure 3. We note that the budget required to get a low relative difference in the large dataset is less than in the size of the graph. Additionally, we confirm that the short random walk estimations of $\hat{p}^3(u,m)$ (where $u \in U$ and $m \in I$) indeed converge to the real $p^3(u,m)$ entry of P^3. This is done using the *total variation distance* $V(P^3) = ||P^3 - \hat{P}^3||_\infty$ and then applied to the small dataset as can be seen in Figure 4.

Small size dataset							
Length of Walk (s)	Evaluation Method	**Truncated hitting times**			**#Hits**	**#Hits * degree**	**$\hat{\mathbf{P}}^{\mathbf{s}}$**
		penalty: Edges $\hat{r}$	penalty: $2\hat{r}$/degree	penalty: Walk length			
3	Metric I	78.98%	79.64%	79.08%	79.17%	80.45%	79.02%
	MetricII@1%	0.1642	0.1592	0.1621	0.1607	0.1760	0.1617
	MetricII@3%	0.3034	0.2962	0.3056	0.3029	0.3300	0.3028
	MetricII@5%	0.3903	0.3896	0.3885	0.3887	0.4286	0.3905
	MetricII@10%	0.5382	0.5545	0.5376	0.5389	0.5812	0.5356
5	Metric I	79.44%	80.12%	79.31%	79.33%	80.47%	70.88%
	MetricII@1%	0.1501	0.1602	0.1472	0.1474	0.1657	0.0905
	MetricII@3%	0.2906	0.2948	0.2916	0.2877	0.3123	0.1947
	MetricII@5%	0.3830	0.3819	0.3780	0.3766	0.4113	0.2755
	MetricII@10%	0.5319	0.5410	0.5253	0.5232	0.5723	0.3959

Table 3: Short Random Walks On the Small Dataset

Large size dataset							
Length of Walk (s)	Evaluation Method	**Truncated hitting times**			**#Hits**	**#Hits * degree**	**$\hat{\mathbf{P}}^{\mathbf{s}}$**
		penalty: Edges $\hat{r}$	penalty: $2\hat{r}$/degree	penalty: Walk length			
3	Metric I	91.83%	93.65%	91.84%	94.90%	94.52%	94.90%
	MetricII@1.0%	0.1769	0.3507	0.1769	0.4681	0.4210	0.4681
	MetricII@3.0%	0.4074	0.5681	0.4074	0.6645	0.6206	0.6644
	MetricII@5.0%	0.5471	0.6766	0.5471	0.7590	0.7281	0.7585
	MetricII@10.0%	0.7461	0.8117	0.7461	0.8657	0.8471	0.8655
5	Metric I	91.04%	93.55%	91.06%	94.37%	94.15%	92.30%
	MetricII@1.0%	0.1495	0.3496	0.1495	0.4347	0.3985	0.3823
	MetricII@3.0%	0.3693	0.5675	0.3693	0.6333	0.6022	0.5821
	MetricII@5.0%	0.5122	0.6751	0.5122	0.7315	0.7103	0.6813
	MetricII@10.0%	0.7157	0.8109	0.7157	0.8474	0.8349	0.8032

Table 4: Short random walks on the large dataset

6. REFERENCES

[1] Anonymous ratings data from the Jester. http://goldberg.berkeley.edu/jester-data.

[2] Last.fm. http://www.last.fm/.

[3] Lg u+ oz store. http://ozstore.uplus.co.kr/.

[4] Movielens data sets. GroupLens Research Lab, Dept. Computer Science and Engineering, University of Minnesota. `http://www.grouplens.org/node/73`.

[5] D. Billsus and M. J. Pazzani. Learning collaborative information filters. In *ICML*, pp. 46–54, 1998.

[6] N. Craswell and M. Szummer Random walks on the click graph. In *SIGIR*, pp. 239–246, 2007.

[7] F. Fouss, A. Pirotte, J.-M. Renders, and M. Saerens. Random-walk computation of similarities between nodes of a graph with application to collaborative recommendation. *IEEE Trans. Knowl. Data Eng.*, 19(3):355–369, 2007.

[8] F. Fouss, A. Pirotte, and M. Saerens. A novel way of computing similarities between nodes of a graph, with application to collaborative recommendation. In *Web Intelligence*, pp. 550–556, 2005.

[9] M. Gori and A. Pucci. Research paper recommender systems: A random-walk based approach. In *Web Intelligence*, pp. 778–781, 2006.

[10] M. Gori and A. Pucci. Itemrank: A random-walk based scoring algorithm for recommender engines. In *IJCAI*, pp. 2766–2771, 2007.

[11] J. L. Herlocker, J. A. Konstan, L. G. Terveen, John, and T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems*, 22:5–53, 2004.

[12] J. Kunegis and S. Schmidt. Collaborative filtering using electrical resistance network models. In *Industrial Conf. on Data Mining*, pp. 269–282, 2007.

[13] S. Lee, S. Song, M. Kahng, D. Lee, and S. Lee. Random walk based entity ranking on graph for multidimensional recommendation. In *RecSys*, pp. 93–100, 2011.

[14] S. Lee, S. Park, M. Kahng, and S. Lee. Pathrank: a novel node ranking measure on a heterogeneous graph for recommender systems. In *CIKM*, pp. 1637–1641, 2012.

[15] S. Lee, S. Park, M. Kahng, and S. goo Lee. Pathrank: Ranking nodes on a heterogeneous graph for flexible hybrid recommender systems. *Expert Syst. Appl.*, 40(2):684–697, 2013.

[16] L. Lovász. Random walks on graphs: A survey. *Bolyai Society Mathematical Studies*, 2:353–397, 1996.

[17] P. Sarkar and A. W. Moore. A tractable approach to finding closest truncated-commute-time neighbors in large graphs. In *UAI*, pp. 335–343. AUAI, 2007.

[18] A. P. Singh, A. Gunawardana, C. Meek, and A. C. Surendran. Recommendations using absorbing random walks. In *NESCAI*, 2007.

[19] L. Zhang, J. Xu, and C. Li. A random-walk based recommendation algorithm considering item categories. *Neurocomputing*, 120(0):391 – 396, 2013.

[20] L. Zhang, K. Zhang, and C. Li. A topical pagerank based algorithm for recommender systems. In *SIGIR*, pp. 713–714, 2008.