The topological face of recommendation

Erwan Le Merrer and Gilles Trédan

Abstract The success of Google's PageRank algorithm popularized graphs as a tool to model the web's navigability. At that time, the web topology was resulting from human edition of hyper-links. Nowadays, that topology is mostly resulting from algorithms. In this paper, we propose to study the topology realized by a class of such algorithms: recommenders.
By modeling the output of recommenders as graphs, we show that a vast array of topological observations become easily accessible, using a simple web-crawler. We give models and illustrations for those graph representations. We then propose a graph-based methodology for addressing an algorithmic transparency problem: recommendation bias detection. We illustrate this approach on YouTube crawls, targeting the prediction of "Recommended for you" links.

1 Introduction

Online recommender systems are tasked to provide users with lists of items similar to the one they are currently accessing (*e.g.*, , video recommendations on YouTube, or items on Amazon). The (item-centered) output of such recommenders is usually seen as a ranked list and accordingly benchmarked using precision or recall metrics [11]. Yet, by allowing users to navigate between item's web-pages through hyperlinks, recommenders implicitly produce graphs [12, 7]. Indeed, the recommendations associated to a displayed item, if accessed (*i.e.*, clicked by the user), also give in turn other recommendations, and so on and so forth; graphs are then a natural way for representing those successive recommendations. Modeling recommenders output as graphs thus allows for leveraging graph theory and the wide spectrum of graph algorithms available for data mining complex networks.

In this light, we first propose the *invivo* characterization of their recommendations by users with the use of a simple web-crawler.

Second, we propose one application for those extracted graphs, related to the field of algorithmic transparency. Some major service providers, such as

Erwan Le Merrer
Technicolor, e-mail: erwan.lemerrer@technicolor.com

Gilles Trédan
CNRS/LAAS, e-mail: gtredan@laas.fr

YouTube, comment on the high level implementation of their recommender, without specifying details that would allow a transparent use by the public [4]. Recently, there has been an increase in the will for accountability of the service provided by those systems, that can be viewed as black-boxes operating in the cloud, and that a user interacts with by providing her profile or by calling API operations [8, 9, 14]. In the setup of the observation by a user of what she gets as recommendations, both the input (her user profile) and the output (tens of recommendations) are very sparse in comparison to the dataset belonging to the service provider for analysis. In this paper, we show as an application that observed graphs of recommendations, despite their sparsity, bring interesting learning.

The contributions of this paper are twofold. *(i)* We propose models for building graphs from recommender system outputs, and study the observability of such graphs. *(ii)* We leverage this approach to propose a methodology for recommendation bias detection. A synthetic model is analyzed and compared to real data extensively collected on the YouTube video recommendation system.

2 Models for graphs of recommendations

In this paper, we focus on item-item recommendation [11], *i.e.*, other items that are presented to a user enjoying or reviewing a given item. Recommendations on a website take the simple form of a set of displayed items, for the user to interact with. We propose to go beyond the collection of this flat item-set, by crawling from each proposed item recursively. The resulting graph structure can provide useful information on the internals of such recommenders. We hereafter formally introduce two forms of such graphs.

Imagine a service provider taking a snapshot of its recommender's system data-structures, so that all operations on those data-structures are frozen and observable at arbitrary time t. From such a snapshot, the recommendations for a given user-item tuple $\langle u, i\rangle$, provided by an arbitrary recommendation algorithm, are observable or computable under the form of *e.g.*, a ranked list of items.

Definition 1 (Graph of recommendations). The directed graph $G_t = (V_t, E_t, W_t)$ is extracted at time t, with V_t the set of available items (nodes) in the system, E_t the set of edges (*i.e.*, the recommendations) connecting some nodes from V_t, and W_t the weight of those edges (*i.e.*, a directed edge (i, j) has a weight $w_{ij} = |\{u \text{ such that } j \in \langle u, i\rangle\}|$). Edges are thus $E_t \subseteq V_t \times V_t$, and gathered from the system data-structures as the union of recommendation lists for every user from user set U_t, at time t: $E_t = \bigcup_{u \in U_t, i \in V_t} \langle u, i\rangle$.

Informally, this graph of recommendations contains the full merged recommendation output for all users, at time t. While observing this structure

over time would provide rich information about the recommender *algorithm*, it is available only to the service provider (as requires direct access to the recommender data-structures.

However, each user u can simply access the web-page of a given item i and observe the set recommendations $V(i)$ she gets for that item. Then, for each recommendation $j \in V(i)$, she can again go to the web-page of item j and observe $V(j)$, therefore recursively crawling her *user-graph of recommendations*, the user-centric counterpart of Definition 1 we now define:

Definition 2 (User-graph of recommendations). The directed graph $G_t(u) = (V_t, E_t)$ is extracted at time t, using the same data-structures as in Definition 1, with the restriction to recommendations to a single given user u: $E_t = \bigcup_{u,i \in V_t} \langle u, i \rangle$.

In practice, a user will certainly limit the depth at which she conducts a recursive crawl from a given item. To that end we introduce $G_t^h(u, i)$ as the subgraph of $G_t(u)$ induced by the h-hop neighborhood of item i for user u.

G_t from Definition 1 is of interest for service providers, for instance for estimating user flows among the proposed videos (*e.g.*, such as in the *random surfer* model over web-pages linked as a graph [2]), while $G_t(u)$ from Definition 2 is of interest for a user observation of her recommendation outputs (as discussed in the sequel).[1]

Practical considerations for observation by a user

While snapshotting data-structures to build G_t or $G_t(u)$ for a specific user u is expected to be relatively straightforward for service operators with full control over their recommender system, this is not feasible for an external observer such as a user of a platform proposing recommendations.

The observation by a user u of her user-graph of recommendation may *e.g.*, be conducted through the platform API or by web-crawling. Because item/user churn and recommendation computation on a large platform is expected to occur quickly, and because a crawl is in essence a sequential operation, such an observation may differ from a system snapshot $G_t(u)$: *(i)* clearly, the worst case for an observer is when in between the access to two items, the system has updated its recommendations, thus possibly triggering another state for the user-graph of recommendations. *(ii)* Yet, for practical implementation reasons of the recommender service (most notably recommendation latency), batch-oriented pre-computations are implemented, rather than on-demand computations [5]; this means that recommendations are updated for instance few times a day, which leaves the observer with a more stable system to observe.

[1] We note that these graph structures and their dynamics relate them to *time-varying networks* and *time-varying graphs* [3]

To acknowledge the potential perturbation induced by the sequential nature of the crawl, we denote $\hat{G}_t^h(u,i)$ the observed user-graph of recommendations. Two strategies can be employed to limit the differences between $\hat{G}_t^h(u,i)$ and $G_t^h(u,i)$:

- crawl as quickly as possible to mitigate recommendation re-computations.
- limit exploration depth using a small h (*i.e.*, local observation around the initial item i), preventing effects from an access to a large set of items.

3 User-graphs of recommendations: first observations

We use the example of video recommendation in this paper. In the mainstream YouTube platform, we start crawling recommended video web-pages from a given video (a popular one from Lady Gaga). The crawling does not appear to be rate-limited, so we could collect tens of thousands of web-pages in the order of a minute. We remark that 19 videos are recommended in each video web-page. We now report some investigations regarding two important parameters of the crawling procedure for building graphs: the temporal aspect (*when* is performed the crawl) and the user dimension (*who* is doing the crawl).

Dynamics of observed user-graph of recommendations

While it is impossible to properly measure the accuracy of $\hat{G}_t^h(u,i)$ compared to $G_t^h(u,i)$ as accessed by the service operator, measuring the dynamics of the recommendation definitely provides information about the temporal stability of recommendations provided by YouTube.

We conducted the following experiment: every hour, over a span of 50 hours, we ran a $h = 3$-hop crawl from a very popular (trending) Lady Gaga video[2], without any other user interaction with YouTube during the process. The outcome is that the number of recommended items in common within two successive crawls is on average 74.7%, with a standard deviation of 13.2%. This indicates that an individual crawl does not result in a full reset of recommended items observed in the next crawl, nor in drastic perturbations over observed recommendations.

Second, we performed a crawl per hour at $h = 3$ from the same video and during one month (33 days), and analyzed on Figure 1 the number of unique vertices in each graph (vcount), the total number of edges (ecount) and the average distance between vertices (meandist). The rolling average over one day is presented as the solid line on Figure 1. All three metrics appear stable over the observation period.

[2] "Lady Gaga's FULL Pepsi Zero Sugar Super Bowl LI Halftime Show — NFL", https://www.youtube.com/watch?v=txXwg712zw4

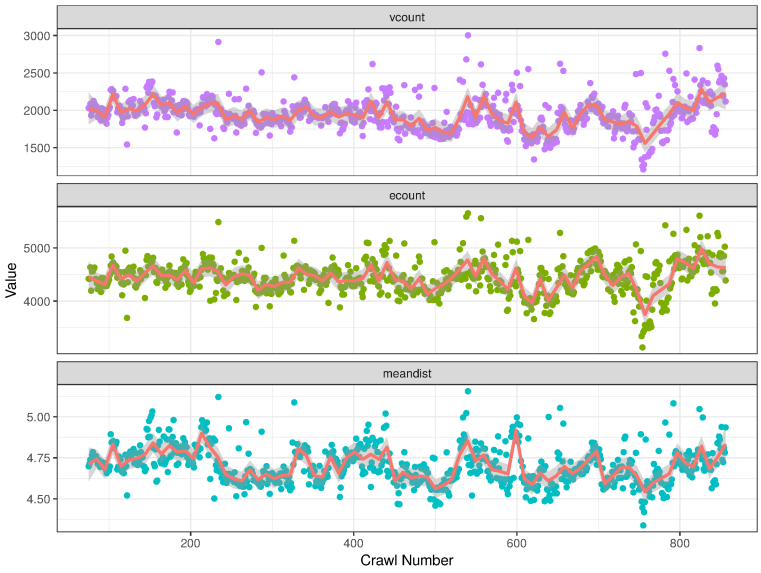

Fig. 1 Observed user-graph statistics for successive hourly crawls, spanning over 33 days.

The personalization of recommendations

How different, given two users u and u', are the graphs $\hat{G}_t^h(u,i)$ and $\hat{G}_t^h(u',i)$? Since YouTube identifies its users using a cookie, the representation given by observed user-graphs of recommendations allows to investigate this question.

We crawled YouTube from the same item, at $h = 4$, and with two users: *(i)* case $\mathfrak{C}$: a user is new to the platform, or *(ii)* $\mathcal{C}$: she is a returning user, and thus has a history on it (recognized by YouTube through the passing of a cookie by our web-crawler). Drastic differences occur when building the graphs for both scenario (the two crawls are separated by only few seconds); $\mathfrak{C}$: $|V(\hat{G}_t^4(\mathfrak{C},i))| = 14,121$ nodes and has $|E(\hat{G}_t^4(\mathfrak{C},i))| = 36,435$ edges. In case $\mathcal{C}$, $|V(\hat{G}_t^4(\mathcal{C},i))| = 8,786$ nodes and has $|E(\hat{G}_t^4(\mathcal{C},i))| = 24,731$ edges.

Both graphs are displayed on Figure 2. Assuming that a recommender provides k recommendations and that each item is only recommended once in $\hat{G}_t^h(u,i)$, we obtain in theory a balanced tree of $n_{tree}^h(k) = \frac{k^{h+1}-1}{k-1}$ nodes. In our context $n_{tree}^4(19) = 137,561$. Therefore, there is a high redundancy of recommendations within both $\mathfrak{C}$ and $\mathcal{C}$. Very interestingly, crawl $\mathcal{C}$ contains around 38% less nodes than $\mathfrak{C}$. This has to be interpreted by the recommender system "knowing" the user, and then trying at each video access to insist in the recommendation of what it thinks is best suited for that user to enjoy. In case $\mathfrak{C}$ (new user), the recommender presents videos related to the start one as well, but also includes in its recommendations videos from different categories (sport, news, ...), in the probable hope to gain knowledge faster about the user by varying its propositions. Those phenomenons are confirmed by analyzing graph structures: a search for main clustered components (through the modularity algorithm with $p = 5.0$) indicates 7 components of size at least

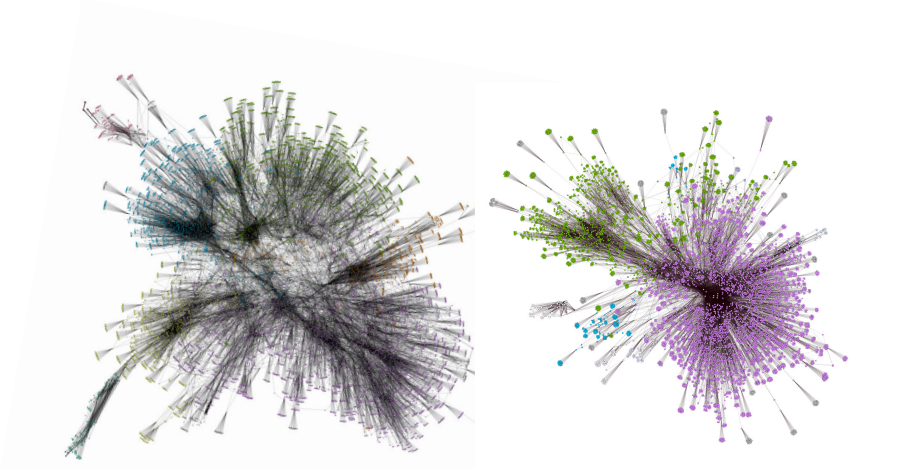

Fig. 2 4-hops graphs from a YouTube video, new user (left) and returning user (right).

1% of the graph size for graph from $\mathfrak{C}$, versus only 3 for case $\mathcal{C}$ (*i.e.*, more precise recommendations lead to fewer clusters of interest for the returning user). Node colors corresponds to components they belong to, on Figure 2. Another key difference is the degree distribution, with 3 nodes having more than 100 in-neighbors for $\mathfrak{C}$, versus 10 in the second case: videos assumed relevant by the system are consistently recommended to the returning user.

An immediate conclusion drawn from the structural analysis of both $\mathfrak{C}$ and $\mathcal{C}$ graphs, is that their topological comparison informs about the degree of knowledge of the system about the observing user, even under a limited exploration scope.

4 An application for graphs of recommendations: detecting recommendation bias with topology analysis

A recently discussed topic is the influence of online medias, and their capacity to shape opinion and user tastes based on item recommendations. Biased recommendations may be proposed by a provider for reasons including: political support, economical (content recommendation versus recommendation of sponsored ads) [1], or serendipity (in the hope to maximize user's journey on the platform through unrelated item discovery) [11]. We propose a technical approach based on user-graphs of recommendations for potential bias identification. We then illustrate it on YouTube crawls.

Recommender Model

We consider recommenders that, given an item $i \in V_t$ (along with some other type of information like the user profile) return a score $s^i_R : V_t \mapsto [0,1]$ typically capturing items similar to i [11]. The output of such a recommender system R is then exploited by a service that selects the subset of best matching items that will get recommended when a user u consults i (typically a ranking operation, as exposed in [4]). Let $R_i(u) \subset V_t$ be this set of recommended items to u at a web-page: $\forall j \in R_i(u), j' \in V_t \setminus R_i(u), s^i_R(u,j) \geq s^i_R(u,j')$. By selecting a recommended item $j \in R_i(u)$, u enjoys j, and in turn gets recommended items similar to j, namely $R_j(u)$.

In this context, the user-graph of recommendations (Definition 2) for u is $G_t(u)$, in which an edge $(i,j) \in E_t \Leftrightarrow j \in R_i(u)$, at time t. In a system that biases recommendations, the user is proposed certain items absent from $R_i(u)$.

Bias in an observed user-graph of recommendations

The service officially exploits the recommender R (such as the one advertised in [4]). The service may add recommendation bias in destination to user u for orienting her navigation among items; this translates into $G_t(u)$ containing biased edges. Let $E_{t,B}$ be this set of biased edges; we thus have $G_t(u) = (V_t, E_t \cup E_{t,B})$, observed as $\hat{G}^h_t(u,i)$.

We now ask the following question: *having access to $\hat{G}^h_t(u,i)$, can user u decide whether a given edge of that graph is biased or not?* Hereafter, we propose a model for addressing this question, that analyzes graph of recommendations from a small-world perspective.

4.1 Identifying biased edges

We propose to study potential bias in recommendation graphs, in relation to the *locality* or not of recommended items. For avoiding a formal - and possibly debatable - definition of bias, we instead state two of its most probable topological consequences on user-graph of recommendations $\hat{G}_t(u,i)$:

Proposition 1 *Biased edges impact graph structure*: if the service bothers to bias recommendations to a user, it is because it effectively impacts that user navigation among items. That is, users do not navigate the same way in an unbiased graph than in a graph $G_t(u)$ containing biased edges. Therefore, the existence of biased edges must significantly change the properties of the observed graph.

Proposition 2 *Recommenders leverage item proximity, and this appears in a graph observation*: most of recommenders exploit some underlying coherence among the items. Collaborative filtering exploits correlations in users'

tastes: if users enjoying a also usually enjoy b, b will be recommended from a and vice versa [11]. This symmetry translates into edge locality in $G_t(u)$, captured by *e.g.*, clustering (as appearing through clusters on Figure 2). Biased edges might not rely on such property, and therefore are less likely to produce locality. We also can argue that if bias actually results in the proposal of usual (*i.e.*, local) items for a user, she is not likely to consider those recommendations as biased.

In practice, machine learning algorithms associate to each item a d-dimensional vector of features; a recommender then for instance relies on k nearest neighbors (kNN) or cosine similarity on those vectors. Recommended items are thus often close-by in the d-dimensional feature space, while we expect biased edges to point to items that are relatively far in the feature space. Note that this observation of the effect of biased edges on the topology may relate to services providing *serendipitous* recommendations [11] (for bringing diversity to a user); this nevertheless arguably constitutes a form of bias w.r.t. user habits.

A small-world perspective

We now propose to identify bias using a parallel to the Watts-Strogatz "small-world" model [15]. In that model, nodes have local connectivity in a given space between them (capturing for instance a geographical proximity), but also have so called long-range links (capturing for instance a familial relationship, loosely related to a geographical proximity). The consequences of these long edges are well known: they drastically impact average path length. We argue that biased links added to the recommender output have the same impact: provided they are *different enough* from the recommender edges in an observed user-graph of recommendations, they will impact the graph structure. To capture this degree of difference, we propose the following model.

A biased recommendation model

Our model considers two recommenders: one "official" R, and one issuing "biased" recommendations, R' (we validate this model experimentally in the sequel). To model the fact that biased recommendations may not be completely unrelated to normal recommendations, we use a tuning parameter $i_{RR'}$ presented hereafter. Let R be a kNN recommendation system producing k_R items per query ($\forall i \in V_t, |R_i(u)| = k_R$), based on the $p_i \in \mathbb{R}^d$ feature vector: $\forall i,j \in V_t^2, s_R^i(u,j) = ||p_i p_j||_2$. Let E_t be the set of produced edges.

In addition to its p_i vector, each item $i \in V_t$ is associated with another d' dimensional feature vector $b_i \in \mathbb{R}^{d'}$, representing "hidden" features (profitability, political support, ...) leveraged to bias recommendation. Biased edge set $E_{t,R'}$ is produced by a kNN recommender with k_B output using b_i. The user-graph of recommendation is then $G_t(u) = (V_t, E_t \cup E_{t,R'})$ in which nodes have an out-degree of $k = k_R + k_B$.

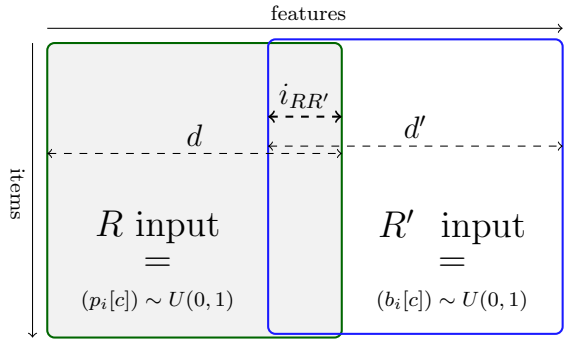

Fig. 3 Illustration of the bias model. The $i_{RR'}$ parameter controls the coupling between the official recommender (R) and the recommender that is source of bias (R').

Features are generated uniformly at random: $\forall i \in V_t : (p_i[c])_{1 \leq c \leq d} \sim U(0,1)$ and $(b_i[c])_{1 \leq c \leq d'} \sim U(0,1)$. However, to vary the amount of bias, the dependency between p_i and b_i is as follows: let $0 \leq i_{RR'} \leq min(d, d')$. We set $\forall i \in V_t, \forall 1 \leq c \leq i_{RR'}, b_i[c] \leftarrow p_i[c]$. That is, if $d = d' = i_{RR'}$, both recommenders R and R' will produce exactly the same results (and therefore equivalent user-graphs of recommendations). On the other hand, if $i_{RR'} = 0$, p_i and b_i are independent, and so are the results of R and R' (see illustration on Figure 3).

4.2 Experiments

We use three YouTube crawls, such as the one presented as $\mathcal{C}$ (Section 3), from a returning user u, and from the same Lady Gaga video (with around one month delay between each of them). In the set of YouTube recommendations at each page, some (around 20%) are tagged with the flag "Recommended for you" (whereas other videos simply display their number of views). For the experiment, we consider those recommendations as part of the biased set we seek to tag (*i.e.*, the ground truth). We configure the model for comparable topology properties: $|V_t| = 8753, k_R = 17, k_B = 2$. We set $d = d' = 5$.

First, we look at the small-world parallel with introduced bias under ***Proposition 1*** on Figure 4; we plot the path length distribution of respectively one $\hat{G}_t^4(u, i)$ observation (said biased) with i the Lady Gaga video, and one $\hat{G}_t^4(u, i) \setminus E_{t,R'}$ (said unbiased): we respectively use the YouTube full crawl, and then remove the "Recommended for you" edges to obtain the unbiased graph.

We note a clear change in the graph properties when biased edges are present: they shortcut many paths, and then cause a more compact distri-

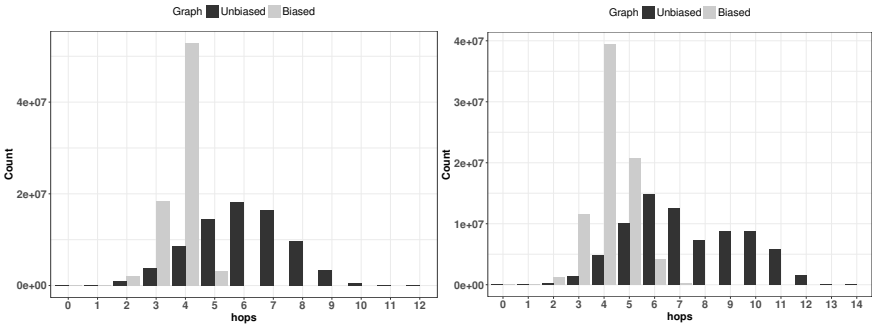

Fig. 4 Distribution of graph path lengths for the experiments: model (left) and a YouTube crawl (right).

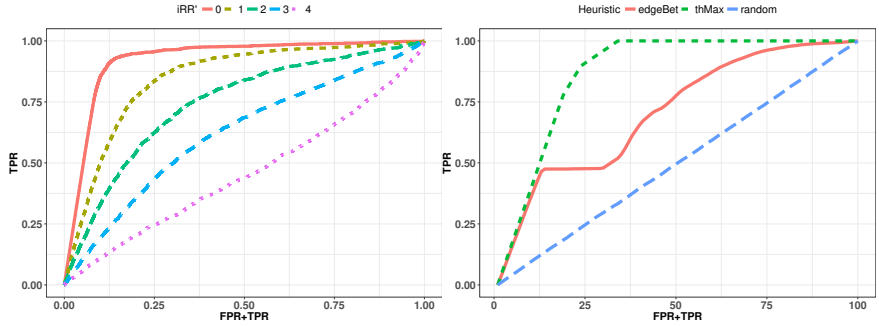

Fig. 5 Feasibility of tagging biased edges: ROC curves for an edge-betweenness tagging algorithm. Model (left) and YouTube crawls (right).

bution of lengths, on both the model and the YouTube crawl. This precise effect of long edges is a well studied feature of small-world graphs [15].

Second, ***Proposition 2*** is examined on Figure 5. For doing so, we run a symmetric edge-betweenness centrality [6] algorithm on $\hat{G}_t^4(u,i)$ graphs, for that metric is aimed at finding topologically important edges (typically linking clusters). We plot the ROC curves, representing the probability of biased edges actually ending-up in sorted top-result of the centrality metric (note that the the YouTube experiment, the plot is averaged over the three crawls). For the model, independent set of edges (iRR=0) indicates the awaited ease to differentiate them, while a bias based on few feature dependencies (*e.g.*, iRR'=1 or 2) also clearly allows for accurate edge tagging. For YouTube, the heuristic is also significantly above a random tagging baseline, close to the maximum possible (thMax) at the beginning of the experiment (*i.e.*, top-ranked edges are indeed all biased). There is then a plateau regime (following top-ranked edges are mistagged), after which accurate tagging progresses smoothly. For both experiments, we conclude that future bias detection algorithms have a clear room for providing accurate results: if one seeks the 4491 biased edges of the dataset, the edge-betweenness algorithm on the YouTube crawl directly allows to identify close to 50% of the biased set, at $x = 12.5\%$.

5 Related Work

The usage of graph representations are common in the literature of recommender systems, for analyzing their **input datasets**, and adapt recommender algorithms accordingly. Notably, authors of [10] propose multiple graph representations of input of a recommender dataset (ratings, users, items): a bipartite graph of users and items, a "social network" of users having consumed the same items, and both the previous graphs with a so called *recommender graph*. Dataset is assumed to be the resulting of users activity in the system following recommendations. They show the approach useful for comparing recommender algorithms as a function of the pairs of users they are connecting a posteriori. The analysis of the **output** of recommenders, possibly at runtime, has been considered on an item-item graph representation, for means of assessing the reachability of items [12] or the navigability of the graphs [7]. We propose in this paper a characterization of the models of graphs that arise from recommendations, and target bias detection as an application.

There is a recent research focus on providing tools capable of assessing the behavior of remote algorithms. XRay [9] proposes a Bayesian approach for inferring which data of a user profile, given as an input, is associated to a personalized ad to that user. Authors propose in [8] a graph theoretic approach to gain understanding on which *centrality* metrics are in use by platforms that offer peer-ranking services. Work in [14] shows that machine learning models can be extracted by a user from the cloud platform were they reside. Regarding recommender systems, paper [13] exposes the users perspective on what they expect from recommendation; interestingly, transparency was already a concern in 2001.

6 Conclusion

There are two main conclusions to this study. First, introduced graph models are interesting tools for analyzing recommender outputs. Graphs of recommendations may be leveraged by service operators to compute general metrics about the consequence of their recommendations to users, in the light of *e.g.*, PageRank applied to the graph of web-pages. User-graphs of recommendations, when observed by users themselves also carry information: they clearly display the item locality a service provides users with; we illustrate this through the clustering of recommendations to a returning user and later make the parallel with small-worlds.

Second, we believe that the representation of user-graphs of recommendations are of important interest for the growing domain of algorithmic transparency: we argue that if a service aims at biasing some recommendations,

the effects will be witnessed on graph topologies. We explored this question by providing a model and by analyzing an extracted graph of recommendations.

These conclusions bring both analytical and algorithmic interest in the scope of personalization transparency. Future work includes generalization to other recommender types (*e.g.*, neural network based), and other applications of graphs of recommendations.

References

1. Bashir, M.A., Arshad, S., Wilson, C.: "recommended for you": A first look at content recommendation networks. In: IMC (2016)
2. Blum, A., Chan, T.H.H., Rwebangira, M.R.: A random-surfer web-graph model. In: ANALCO (2006)
3. Casteigts, A., Flocchini, P., Quattrociocchi, W., Santoro, N.: Time-varying graphs and dynamic networks. In: H. Frey, X. Li, S. Ruehrup (eds.) ADHOC-NOW (2011)
4. Covington, P., Adams, J., Sargin, E.: Deep neural networks for youtube recommendations. In: RecSys (2016)
5. Davidson, J., Liebald, B., Liu, J., Nandy, P., Van Vleet, T., Gargi, U., Gupta, S., He, Y., Lambert, M., Livingston, B., Sampath, D.: The youtube video recommendation system. In: RecSys (2010)
6. Girvan, M., Newman, M.E.J.: Community structure in social and biological networks. Proceedings of the National Academy of Sciences **99**(12), 7821–7826 (2002)
7. Lamprecht, D., Strohmaier, M., Helic, D.: A method for evaluating the navigability of recommendation algorithms. In: International Workshop on Complex Networks and their Applications (2016)
8. Le Merrer, E., Trédan, G.: Uncovering influence cookbooks : Reverse engineering the topological impact in peer ranking services. In: CSCW (2017)
9. Lécuyer, M., Ducoffe, G., Lan, F., Papancea, A., Petsios, T., Spahn, R., Chaintreau, A., Geambasu, R.: Xray: Enhancing the web's transparency with differential correlation. In: USENIX Security Symposium (2014)
10. Mirza, B.J., Keller, B.J., Ramakrishnan, N.: Studying recommendation algorithms by graph analysis. Journal of Intelligent Information Systems **20**(2), 131–160 (2003)
11. Ricci, F., Rokach, L., Shapira, B., Kantor, P.B.: Recommender Systems Handbook, 1st edn. (2010)
12. Seyerlehner, K., Knees, P., Schnitzer, D., Widmer, G.: Browsing music recommendation networks. In: ISMIR (2009)
13. Sinha, S., Rashmi, K.S., Sinha, R.: Beyond algorithms: An hci perspective on recommender systems (2001)
14. Tramèr, F., Zhang, F., Juels, A., Reiter, M.K., Ristenpart, T.: Stealing machine learning models via prediction apis. In: USENIX Security Symposium (2016)
15. Watts, D.J., Strogatz, S.H.: Collective dynamics of'small-world'networks. Nature **393**(6684), 409–10 (1998)