An Homophily-based Approach for Fast Post Recommendation in Microblogging Systems

Quentin Grossetti, Camelia Constantin, Cédric Du Mouza, Nicolas Travers

An Homophily-based Approach for Fast Post Recommendation on Twitter

Quentin Grossetti
CNAM, UPMC
Paris, France
quentin.grossetti@upmc.fr

Camelia Constantin
University Pierre et Marie Curie
Paris, France
camelia.constantin@lip6.fr

Cédric du Mouza
CNAM
Paris, France
dumouza@cnam.fr

Nicolas Travers
CNAM
Paris, France
nicolas.travers@cnam.fr

ABSTRACT

With the unprecedented growth of user-generated content produced on microblogging platforms, finding interesting content for a given user has become a major issue. However due to the intrinsic properties of microblogging systems, such as the volumetry, the short lifetime of posts and the sparsity of interactions between users and content, recommender systems cannot rely on traditional methods, such as collaborative filtering matrix factorization. After a thorough study of a large Twitter dataset, we present a propagation model which relies on homophily to propose post recommendations. Our approach relies on the construction of a similarity graph based on retweet behaviors on top of the Twitter graph. Finally we conduct experiments on our real dataset to demonstrate the quality and scalability of our method.
Keywords: Recommender System; Collaborative Filtering; Microblogging Systems

1 INTRODUCTION

During the last decade, several microblogging platforms have emerged such as `Twitter`, `Pinterest`, `Instagram`, `Weibo` or `Tumblr`. These platforms rely on the same paradigm: users follow each other's and share content. They have their specific audience and features but all have encountered unprecedented growth. This growth tends to make microblogging platforms overcrowded and users to encounter difficulties to keep up with all the available content. For instance, in 2017, 500 million messages were published every day on `Twitter`. Finding relevant messages to recommend in real time is a real challenge of prime interest for these platforms. Indeed, effectively recommending fresh publications leads to higher engagement from users which is crucial for a social media service [12].

Many recommendations methods have been proposed in the literature. Some works propose to extract features from items to recommend them to suitable users like [23]. However this approach provides poor results on microblogging platforms due to the short length of the messages (140 characters for a tweet), and a broad variety of content: video, picture, sound etc. Other methods based on collaborative filtering try to capture similarity between users based on "co-liked" items and to use this similarity to determine recommendations [30]. Among collaborative filtering methods, matrix factorization [20] and social trust models [25] are very popular. If these methods provide relevant recommendations to end-users, they can not scale up with the huge amount of items produced by a platform like `Twitter` despite existing optimization techniques [26]. For instance the similarity matrix size will be of 1 000 billion of messages with millions of users, and will permanently grow up, so in addition to storage issues (even if this is a sparse matrix), it requires constant and costly computations.

To face this issue, `Twitter` has developed its own recommender system named *GraphJet* [32]. This method relies on a bipartite graph built on users interactions with messages and computes random walks. By starting these walks from a query user, it is possible to compute a list of personalized recommendation at low cost [32]. However, due to its random walk-based computations, *GraphJet* tends towards recommending mostly popular messages. Even if many users tends to focus on popular tweets, this approach reduces chances to recommend more specific content with similar interests in the neighborhood.

Based on a thorough analysis of a real `Twitter` dataset of 2.2M users extracted in 2015, with a special focus on the homophily concept, we propose in this article an original approach which achieves a good trade-off between relevance, scalabity and freshness of recommendations. Our intuition is that recommendation relevance can be enhanced by understanding how people with similar profiles are interconnected. Our proposal relies on *SimGraph*, a similarity graph which links users with relevant-based edges based on common retweets, along with a scalable propagation model.

In a nutshell, the main contributions of this work are:

(1) a micro-blogging analysis which focuses on retweets and homophily characterization;
(2) a novel method to drastically reduce the cost of collaborative filtering methods relying on homophily and the construction of a similarity graph *SimGraph*
(3) a convergent and scalable propagation algorithm enhancing recommendations with transitivity along with optimization technics;
(4) a thorough experimental comparison between *Collaborative Filtering*, *GraphJet*, a *Bayes Inference Model* and *SimGraph*, focusing on recommendation quality, processing time and robustness over time.

The rest of the paper is organized as follows. Section 2 is a review of the related works. Section 3 provides a thorough analysis of a `Twitter` dataset with a focus on homophily, which leads to our similarity graph and propagation model in Section 4. Our optimized propagation algorithm is presented in Section 5 and compared to other solutions in experiments in Section 6. Finally, Section 7 concludes the paper.

2 RELATED WORK

Since their appearance in the beginning of the 1990s at Xerox [11], recommender systems have been extremely popular and are used to recommend a large variety of objects such as : music [29], movies [27], news [9], recipes [8], etc. Different types of recommending methods emerged: content-based, collaborative filtering, graph-based, bipartite networks or deep learning.

Content-based methods aim at describing both profiles and items in order to provide recommendation [23]. The strength of content-based methods lies in the capacity to provide recommendations without requiring any feedback from users by extracting tendencies of collected data. In the context of microblogging, the content itself is poor with small-size tweets, links or multimedia, even if some works on `Twitter` targeted this approach [18]. However, content-based models generally suffer from overspecialized recommendations [1].

Collaborative Filtering (*CF*), on the opposite, combines both content and user interactions in order to produce recommendations. *CF* is generally represented as a matrix resolution problem and provides relevant recommendation. Lot of works focus on sparseness [19], on scalability [26] and on cold-start problems [2]. Matrix factorization methods transform both user and item vectors to a latent factor space, where similarities between items and users are generated by lower-dimensional hidden factors. However in the `Twitter` context, matrix factorization is not scalable due to the matrix size and continuous growth, but also it does not allow to consider the freshness of the recommended items.

Some other proposals exploit the social network and weight edges to quantify correlations between users. Jiang et al. [17] mix the social network with individual preferences into a matrix factorization model and improve recommendations. A broad range of social recommender systems have a similar approach like [33]. In Jamali et al. [16], authors present *SocialMF* to also enrich matrix factorization with the network of users. However the size of the `Twitter` matrix (more than 100 billions values) and its high growth (500 millions of messages/day) make these methods hard to exploit. Many works on trust networks have been done with explicit information, like in SoRec [24] on Epinion[1] producing and factorizing probabilistic matrices. However, the relationships between users on microblogging platforms is very heterogeneous and can carry many different meanings. Consequently it can not be really considered as a trust relationship.

The bipartite network model is another popular approach in recommendation systems. For instance `Twitter` proposes in [32] *GraphJet*, a random walk approach on user and post graph. Even though this method is particularly efficient, it only focuses, for scalability purposes, on the freshness of interactions. In our point of view, *GraphJet* by missing to exploit older interactions reduce the complexity of users profiles. Moreover, we will see in the experiments that this solution mainly addresses the hot-topic recommendations.

There exist also other works on `Twitter` for recommending hashtags [10] or users [6], or for filtering messages in the timeline [34]. But very few address the post recommendation problem.

Finally, deep learning methods on huge datasets such as `Twitter` [28] or `Youtube` [7] process neural networks to connect many dimensions such as similarity matrix, content features,

[1]http://www.epinions.com/

# nodes	2.2M
# edges	325.5M
# tweets	3,002M
avg. out-deg.	57.8
avg. in-deg.	69.4
max out-deg.	349K
max in-deg.	185K
diameter	15
avg. path length	3.7

Table 1: Main features of the Twitter dataset

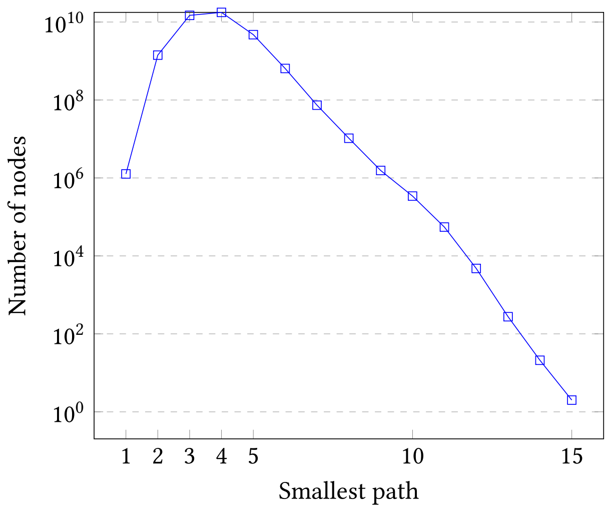

Figure 1: Twitter smallest paths distribution

user feedback. Even if the efficiency of such systems is promising, it faces scalability and dynamicity challenges for microblogging platforms. As a matter of facts, it currently uses past content to extract interesting items from user timelines but does not support real-time recommendation of posts from the whole network.

3 MICROBLOGGING ANALYSIS

We introduce here the main characteristics of our `Twitter` dataset and the different experiments we performed to characterize user behaviors. We discuss the main results of our data analysis and their implication for our recommendation model.

To build our dataset we first extracted a connected component from the graph from Kwak et al. [21]. Then for each node of this subgraph we updated every information in 2015 thanks to the Twitter API [2]. More precisely, for each account we collected the incoming edges (followers), outcoming edges (followees) and all the tweets published by the associated account. Observe that due to the API limit we only retrieved the last 3,200 messages (maximum) for each account. Table 1 summarizes the main features of the dataset. With more than 2 million users and 3 billion messages we have a mean number of 1,375 published tweets per user. The original Kwak connected component contained 125M of edges, so its connectivity has almost doubled in 6 years. The average path length for our graph is 3.7 which is very close to the 4.1 found by Kwak et al. (Figure 1). Likewise, the diameter (longest shortest path between two nodes) for their graph is 18 which is very close to the one we obtained (15). We also observed that our

[2]https://dev.twitter.com/rest/public

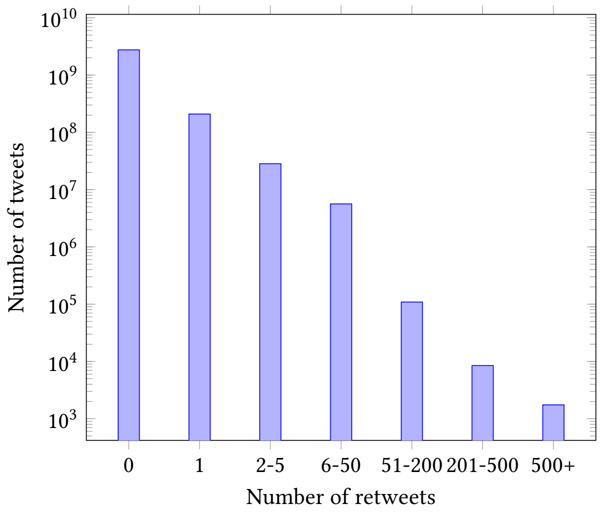

Figure 2: Distribution of the number of retweets per tweet

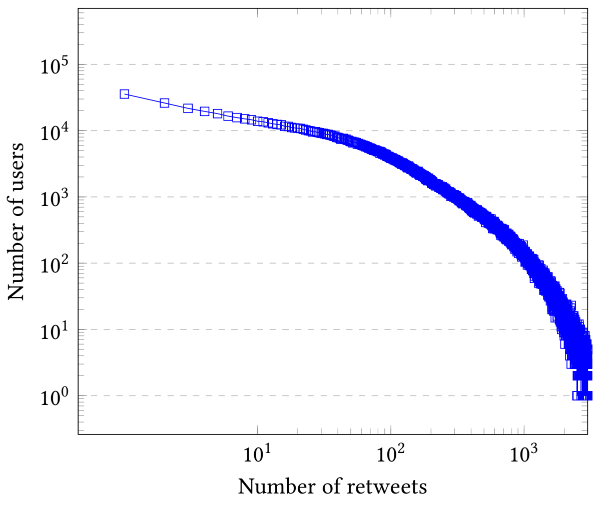

Figure 3: Number of retweets per user

in/out-degree distributions (*i.e.* number of followers/followees per user) are close to the ones found in previous `Twitter` graph characterizations like [22, 35]. We conclude that we succeed in building a representative subgraph of `Twitter` along with the associated tweets published by the different accounts.

Based on our representative dataset we perform a set of experiments to analyze the retweet activity.

3.1 Retweet characterization

Our objective is to recommend relevant tweets to users. Retweet is the mean to express interest for a piece of information, which can indifferently be positively or negatively perceived. The study of tweet propagation through retweets is consequently of primary interest.

3.1.1 Retweet behavior. First, we analyzed the popularity of the retweet behavior. Figure 2 shows the number of retweets for a given message. A large majority of messages are never retweeted (≈90% of the tweets), or only a few times, with barely 2-3 retweets (≈2%). Very popular messages are extremely rare: messages with more than 50 retweets represent less than 0.005%. These results are consistent with the study of Kwak [21] which underlines that a very large majority of the messages are never retweeted or retweeted only once. From a recommender system point of view, tweets which are retweeted more than 4-5 times must get a significant weight since they are popular. Observe that the network itself promotes the propagation of this type of messages (social network effect). However being able to recommend a recent tweet with a small number of retweets (less than 5) is a real challenge. This is one of the objective of our proposal.

When analyzing whether retweeting is a common behavior, we observe that only a small number of users produce many retweets (see Figure 3). The majority of users perform between 10 and 100 retweets. The average number of retweets per user is 156 while the median is 37.5 and we observe the traditional power law distribution where few people gather all the retweets. From the point of view of recommender systems, the main difficulty lies in users with very few retweets (or even none) who represent a large part of the users (a quarter of users have never retweeted). For these users, methods which rely on collaborative filtering would be unable to provide recommendations.

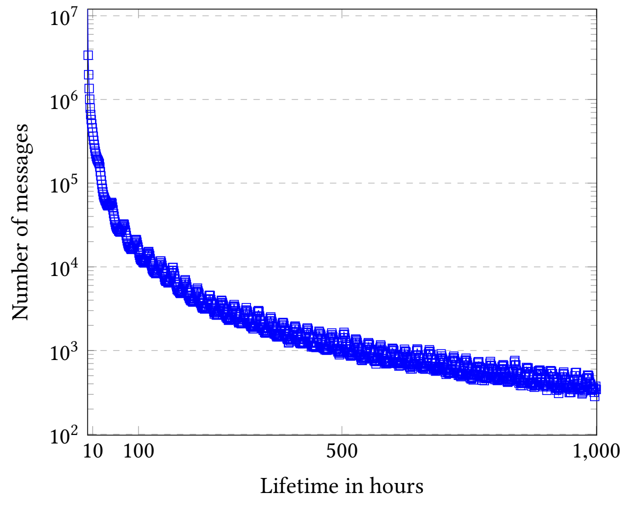

Figure 4: Lifetime of a tweet

3.1.2 Lifetime of a tweet. Recommender systems must avoid recommending a tweet which is "outdated", *i.e.* which won't be propagated anymore because it no longer interests anyone. We study here the lifetime of a tweet that we measured as the time span between the publication of the initial message and the last time it was retweeted. Only messages which were retweeted at least once are considered in this experiment. The results are shown in Figure 4. We observe that a large part of the messages "die" (so are no longer retweeted) before reaching one hour (40%). 90% of the messages die before 72h (three days). It's very uncommon for a message to be retweeted beyond that point. Therefore our results differ from the ones reported by Kwak et al. [21] which notice that 10% of the retweet activity takes place one month after the publication of the message. Indeed, `Twitter` still was, in 2009, a recent system and less content was created, thus it was more easy to retweet messages older than a few weeks. In 2017, with more than 500 million new tweets published every day, freshness of the tweets has become a central criterion of propagation.

We conclude from this analysis that users mainly share fresh information and the tweets to consider in our recommender system are the messages that reach at least 2 retweets. Moreover, even for these messages, we do not need to compute their score after 72h because after this limit they become irrelevant for recommendation.

3.2 Homophily study

Collaborative filtering methods rely on a list of *similar* users to determine recommendations. We study in this section the evolution of similarity scores between users with respect to their distance in the Twitter network. We adopt a Jaccard similarity measure with a slight adjustment to take into account the popularity of the tweet as advised by Breese et al. in [4]: the less popular a retweeted post by two users is, the closer these users are. The rationale for relying on retweets to measure the similarity between users, as explained before, is that retweets are the only way a user expresses explicitly an interest about a piece of information published.

Definition 3.1 (User similarity measure). The similarity of two users is defined as:

$$sim(u, v) = \frac{\sum_{i \in L_u \cap L_v} \frac{1}{log(1+m(i))}}{|L_u \cup L_v|} \quad (1)$$

where $sim(u, v)$ stands for the similarity score between a user u and a user v. L_u is the profile of user u estimated through the set of all the tweets he retweeted. $m(i)$ is the popularity of the tweet, measured here as the number of times it has been retweeted.

Based on this measure we study the homophily between users. Homophily corresponds in social media to the tendency for people to be connected with people sharing the same interest. This effect has been studied on demographic dimensions (age, gender, political orientation) for example by Colleoni [5] or Zamal [37]. They show how one could predict a user demographic based on his neighborhood. Topical homophily also has been studied by Lerman [15], that shows how people who are topically more similar are also more likely to be connected. Bhattacharya et al. [3] present similar results. With our analysis we study the homophily phenomenon in a different perspective.

Because of computation costs, we limited our study to 2,000 users randomly selected from our dataset, checking that they retweeted at least a defined number of posts. For each of these users, we computed the length of the shortest path connecting them to all other users. The results are shown in Table 2. We see that only 5.96% of user pairs display a similarity score over 0 and are directly connected through the network. However this small number of users pair presents the highest mean similarity score with 0.0056. When exploring at distance 2, to reach the followees of the followees, we note that the mean similarity score of 0.0021 is still higher than the average value (0.0019), and they represent almost 38% of the user pairs. Most of the users (51%) with a similarity score not null are at distance 3 from each other. However their similarity score is lower, 0.0017. Moreover, we observe that, according to the topology of the `Twitter` network (Figure 1), a distance of 3 corresponds almost to the entire network.

Table 3 completes this experiment by studying the link between position in the Top-N of most similar users and their distance to the user. For each of the 2,000 users from the previous experiment, we collect the 5 users with the highest similar scores. We then compute the average shortest distance between a user

Distance	Nb of users	Perc.	Average similarity
1	19,163	05.96%	0.0056
2	121,857	37.91%	0.0021
3	166,633	51.84%	0.0017
4	12,070	03.76%	0.0018
5	297	00.09%	0.0016
6	6	00.01%	0.0019
Impossible	1,396	00.43%	0.0023

Table 2: Evolution of the similarity score through distance in the network

		Distances distribution (%)			
Rank	Average Distance	1	2	3	4
1	1.65	53.30	28.20	16.65	1.45
2	1.78	43.70	34.50	20.50	1.05
3	1.88	37.99	36.04	24.37	1.35
4	1.97	33.18	36.99	27.68	1.70
5	1.99	32.01	37.93	28.20	1.56

Table 3: Link between distance in the network and position in the *Top-N* ranking

and each user from his Top-5. We observe that the user at first rank, *i.e.*, the most similar user, is directly connected to the user in 53% of the cases. The average score decreases when going down in the user rank. For instance the user ranked at the fifth position is directly connected to the user in only 32% of the cases. This study of the top-5 reveals that when considering users at a distance 1 added with users at distance 2 we capture 70-80% of the most similar users.

From this experiment we conclude that relying on the "strong" homophily, *i.e.* direct connection between users, is not enough because they represent a very small subset of the set of similar users. On the other hand we observe a "soft" homophily corresponding to users having a good similarity score but located at distance 2. These users which represent around 37% of users with a not null similarity scores, must consequently be considered by recommender systems to determine meaningful recommendations.

4 OUR MODEL

Based on our previous analysis we propose a propagation model relying on homophily to build post recommendation. Our approach relies on the construction of a similarity graph on top of the `Twitter` graph. The basic idea is to exploit the fact that highest similar users are at a distance lower or equal to 2. Thus we can use an exploration of the network to drastically reduce the amount of computed similarities.

4.1 Similarity graph

Our experiments about the homophily enlighten that the natural homophily in Twitter, which is translated by a "follow" link, is not sufficient to detect users of interest while considering all nodes at a distance up to 2 allows to capture most similar users. So, we decide to perform a 2-hop exploration from each node (user) in the Twitter graph. The set of reachable nodes for a node u is named the 2-hop neighborhood, denoted $\mathcal{N}_2(u)$. Then we compute for each node v in $\mathcal{N}_2(u)$ the similarity score of v with all the nodes from $\mathcal{N}_2(u)$. For each node w with a similarity greater

	SimGraph
Nb of nodes	1,149,374
Nb of edges	4,950,417
Mean Similarity Score	0.0078
Mean out-degree	5.9
Diameter	21
Mean smallest path	7.5

Table 4: ***SimGraph*** **characteristics**

than a chosen threshold τ, we create an oriented edge $e(u, w)$ in our similarity graph.

So formally the definition of the similarity graph is:

Definition 4.1 (Similarity graph). Consider the Twitter graph $G(V, E)$ where V denotes the set of vertices and E the set of edges, and a given similarity threshold τ. The similarity graph $SimGraph(V', E')$ is defined as:

$$\begin{cases} V' \subseteq V \\ e(u, v) \in E' \Rightarrow u \in V \wedge v \in \mathcal{N}_2(u) \wedge sim(u, v) \geq \tau \end{cases}$$

We report in Table 4 the main characteristics of the similarity graph for our Twitter dataset. First, we observe that around half of the users from the original Twitter dataset are not present in the similarity graph. The reason is that many users either don't retweet any message or nobody has retweeted the same messages as this user. This issue is very close to cold-start problems and we won't address in this paper the issue of recommending tweets for users who are not present in the similarity graph.

However, we could consider an approach similar to the one used in *GraphJet* [32] using the neighborhood's computed recommendation of cold start nodes to partially solve this issue. Therefore it seems possible without too much effort to recommend items also for cold-start users.

Another characteristic of the similarity graph is a more uniform in-degree distribution than the original graph, with few nodes having a large number of incoming edges. This property is particularly interesting since in the opposite situation it would have led to some users gathering all the influence in the similarity network. Paths between two nodes in our similarity network are now much longer than the original `Twitter` network since the diameter of the similarity graph is now 21 and the average smallest path was doubled from 3.7 to 7.5. We observe that we are still in a small world network context according to the definitions expressed in the work of Schnetrler [31]. Basically, this construction of a similarity graph, relying on 2-hop exploration in the `Twitter` graph could be seen as a dimension reduction process.

4.2 Propagation Model

The sparsity of user interactions on data is generally considered a major bottleneck for a successful execution of a collaborative filtering recommender system. As we've seen before, in microblogging context, this sparsity issue is important. In their work, Huand et al. [14] use transitivity in a collaborative filtering model to fight the sparsity problem. Considering a similar approach, we propose a propagation model on top of our similarity graph. Propagation is an efficient way to face the sparsity since it allows to transmit content from a user u to a user v while there exists no link (*i.e.*, edge in the underlying graph) between them. Intuitively we consider that if a user w has interests similar to u

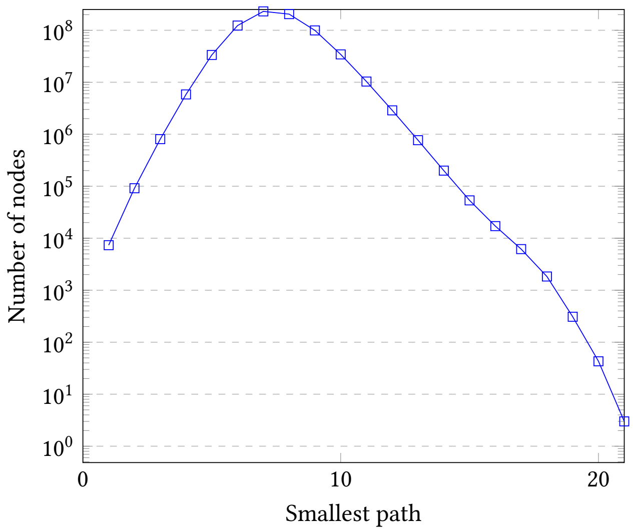

Figure 5: ***SimGraph*** **smallest path**

leading to several retweets, the content from u that we recommend to w should also be recommended to v if v is similar to w.

The only action we can collect to capture interests from a user for a tweet is the retweet action to share a piece of information. Consequently we assume here that liking a tweet is similar to sharing it (retweet). Therefore the words "like" and "retweet" are interchangeable. We estimate the sharing probability, *i.e.* the probability that a user u likes a tweet t as:

Definition 4.2. Sharing probability The probability that a user u likes (and shares) a tweet t is:

$$p(u, t) = \frac{\sum_{v \in F_u} p(u \leftarrow v, t)}{|F_u|}$$

where F_u denotes the set of influential (similar) users for u (*i.e.*, those connected by outcoming edges in the similarity graph) and $p(u \leftarrow v, t)$ is the probability that u likes t determined consistently by the behavior of his influential user v. This probability is estimated as:

$$p(u \leftarrow v, t) = p(v, t) \times sim(u, v)$$

Example 4.3. Consider the similarity graph of Figure 6 with 5 nodes (u, v, w, x, y). An edge $u \longrightarrow v$ expresses the fact that v is an influential user of u. Edges values correspond to the similarity score $sim(u, v)$. Assume that user x liked/shared the tweet $t1$, so the probability that x like $t1$ is $p(x, t1) = 1$. Consequently the score of $p(w, t)$ is:

$$p(w, t1) = \frac{\sum_{v \in F_w} p(w \leftarrow v, t)}{|F_w|} = \frac{0 \times sim(w, y) + 1 \times sim(w, x)}{2} = 0.25$$

However since $p(w, t1)$ changes, all probabilities which depend on the value of $p(w, t1)$ must be updated in turn.

5 PROPAGATION ALGORITHM

We introduce in this section our propagation algorithm which allows to recommend new content to users based on their similarity. We also present its convergence property and optimizations.

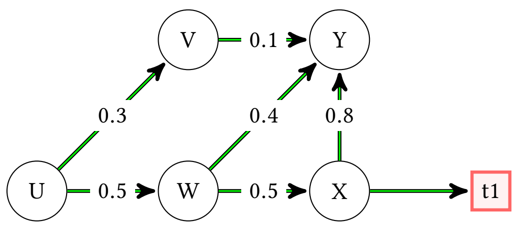

Figure 6: Example of similarity graph

5.1 Iterative algorithm

We perform our iterative propagation algorithm each time a tweet t is retweeted. We discuss different optimizations in Section ?? which exploit our observations regarding the popularity and the lifespan to avoid performing the propagation for each retweet.

So assume that we have a similarity graph $SimGraph = (V, E)$ with V the set of vertices and E the set of edges, and that a tweet t is retweeted by a user. Algorithm 1 presents our propagation algorithm. We denote by D the set of users that have already retweeted t. For each $v \in D$, the probability $p(v, t)$ is consequently 1 (line 3). For all other users $u \in V \setminus D$ we consider that the initial probability to retweet this tweet is $p(u, t) = 0$ (line 4). During an iteration, we compute for all the users u in $V \setminus D$ the probability $p(u, t)$ based on the probabilities from his followees, *i.e.* $p(v, t)$ for all $v \in F_u$ (line 10). So for iteration k we compute our probabilities based on the those which were computed at iteration $k - 1$. Observe that users $v \in D$ have a probability of 1 which is not re-computed during the iterative algorithm. The algorithm stops when no probabilities change during an iteration (line 7).

Algorithm 1: Propagation algorithm

```
input  : Similarity graph SimGraph = (V,E), a tweet $t$ and a
         set of users $D$ who retweet it
output : the set of vertices with their retweet probability
1  Initialization;
2  foreach $u$ in $V$ do
3      if $u \in D$ then $p(u, t) = 1$ ;
4      else $p(u, t) = 0$;
5  end
6  convergence = false ;
7  while convergence = false do
8      convergence = true ;
9      foreach $u$ in $V \setminus D$ do
10         $p'(u, t) = (\sum_{v \in F_u} p(v, t) \times sim(u, v))/|F_u|$
11         if $p(u, t) \neq p'(u, t)$ then convergence = false;
12         $p(u, t) = p'(u, t)$ ;
13     end
14 end
15 Return $\forall u \in V, p(u, t)$ ;
```

Example 5.1. Consider our previous example in Figure 6. After that user x liked/shared the tweet $t1$, we propagate this action on the similarity graph and we first update the score of $p(w, t)$ since x is an influential user of w. Since the value of $p(w, t)$ changed and is now 0.25 we must update in a second iteration the probability of the users influenced by w, so here only u.

$$p(u, t1) = \frac{\sum_{v \in Fu} p(u \leftarrow v, t))}{|Fw|} = \frac{0 \times sim(u, v) + 0.25 \times 0.5}{2} = 0.0625$$

Since there is no users influenced by u the algorithm stops.

Therefor this algorithm produces for an incoming tweet a probability score for every users in the propagated sub-graph. Then, the recommendations of a user is based on the top-k tweets sorted on probability. We will study in Section 6.2 the impact of the size (k) on the quality of the recommendations compared to competitors.

5.2 Linear system formulation

Consider we have n users $(u_1, u_2, ..., u_n)$ in the similarity graph, our propagation model consists in resolving the following linear system with n equations:

$$\begin{cases} a_{11}p_{u_1} + a_{12}p_{u_2} + ... + a_{1n}p_{u_n} & = & b_1 \\ a_{21}p_{u_1} + a_{22}p_{u_2} + ... + a_{2n}p_{u_n} & = & b_2 \\ \cdots & = & \cdots \\ a_{n1}p_{u_1} + a_{n2}p_{u_2} + ... + a_{nn}p_{u_n} & = & b_n \end{cases}$$

which can be written under a matrix product $Ap = b$ with:

- vector b which is the initial state vector, with $b_i = 1$ if u_i has retweeted (shared) the message and 0 otherwise;
- vector p is the solution vector, containing the probabilities after propagation of any user to like the tweet;
- the similarity matrix A which is defined as:
  $\forall i \leq n, \forall j \leq n,$

$$a_{i,j} = \begin{cases} 1 & if\ i = j \\ \frac{-sim(u_i,u_j)}{|F_{u_i}|} & if\ (u_i, u_j)\ is\ an\ edge\ in\ the\ SimGraph \\ 0 & otherwise \end{cases}$$

5.3 Convergence property

Incremental resolution methods such as Jacobi, Gauss-Seidel or successive over-relaxation (SOR) can be used to solve such a linear system $Ap = b$. A necessary and sufficient condition to ensure convergence for this incremental resolution method is that the matrix A is diagonally dominant. This condition is fulfilled here because $\forall u, v\ sim(u, v) \leq 1$, so $\sum_{j \neq i} |a_{ij}| < \frac{1}{F_u} \times \sum_{j \neq i} 1 = 1$. Since $|a_{jj}| = 1$, we conclude that $|a_{jj}| \geq \sum_{j \neq i} |a_{ij}|$ for all i, so A is diagonally dominant.

Considering the Jacobi method of resolution, the convergence speed is bound by the matrix norm $||A||$. However, this value is dependent of the matrix's values, and therefore cannot be known theoretically. We conducted an experimental study on our dataset and show that the convergence of our model is bound to $||A|| = 0.91$ - the worst case scenario. The unpredictable or bad convergence speed of our model led us to apply several optimizations to guarantee a fast convergence and therefore a fast computation.

5.4 Propagation algorithm optimizations

We propose and test several optimizations for the propagation algorithm which significantly improve its performance.

Propagation thresholds. A first optimization consists in setting a static threshold β to decide whether a score updated at an iteration k should be transmitted to the followees at the iteration

$k + 1$. So when $p(u,t)^n - p(u,t)^{n-1} < \beta$ the user u does not propagate its score to his followees for any following iteration.

We extend this traditional approach by proposing a dynamic threshold based on our analysis about the evolution of the tweet popularity and its lifespan. Indeed the probabilities computation for a popular message requires a long time in order to reach convergence. Moreover, this tweet is likely to be sent to a large part of our similarity network. Oppositely, a tweet which has not been retweeted many times because it has just appeared in the system leads to faster computations. Most recommender systems focus on already-popular tweets. So, by setting a dynamic threshold which favors tweets less popular because they have just appeared, we could recommend them earlier than other recommender systems, and at lower computing costs. Precisely we define our dynamic threshold $\gamma(t)$ for a tweet t as:

$$\gamma(t) = \frac{(m(t))^p}{k^p + (m(t))^p}$$

where $m(t)$ denotes the popularity of the message t which can be measured as the number of retweets, for instance. k and p are fixed parameters and are superior to 0, they are used to fit properly the distribution of popular items. Basically $\gamma(t)$ is bounded between $[0, 1]$ and close to 0 when few people have shared t, and close to 1 when the message is considered popular. However other dynamic thresholds could be considered.

Postponed computation. We propose another optimization based on the time frame update. It consists in starting the propagation process not at each time a new retweet happens on a message but after time interval δ depending on the account activity. For instance, assume that a message is very popular with dozens of retweets per minute. We could decide to wait 10 minutes before executing the propagation computation. On the opposite a very unpopular message can wait few hours before launching the propagation process.

6 EXPERIMENTS

We evaluate the *SimGraph* model along with our propagation algorithm. We first detail the experiment protocol used with our `Twitter` dataset to measure the quality of generated recommendations. Then we present our different results and compare them with other recommendation methods.

6.1 Experimental Setup

Our experiments are performed on a Linux machine with 512GB of RAM memory and 80 Intel(R) Xeon(R) CPU E7-4830 v2 @ 2.20GHz. We use Java Openjdk 1.8 for all our implementations. To compare our method with several baseline solutions, we implement a *Bayes* Inference Model used for recommendation [36], a standard collaborative filtering method (*CF*) [13] and *GraphJet* algorithm [32] which is currently used by `Twitter`. The rationale for the choice of these different competitors is that our method could be seen as a combination of probabilistic and collaborative filtering approaches. Regarding the Bayesian inference model we tweak it slightly to consider only the binary feedback of `Twitter` (liking or doing nothing) instead of ratings from 1 to 5. Moreover due to computational issues it is necessary to define a threshold in the Bayesian probabilities computation to stop the costly process. The implementation of the *GraphJet* method is based on `Twitter` source code[3]. All experiments are performed on the real `Twitter` dataset introduced in Section 3.

[3]https://github.com/twitter/GraphJet

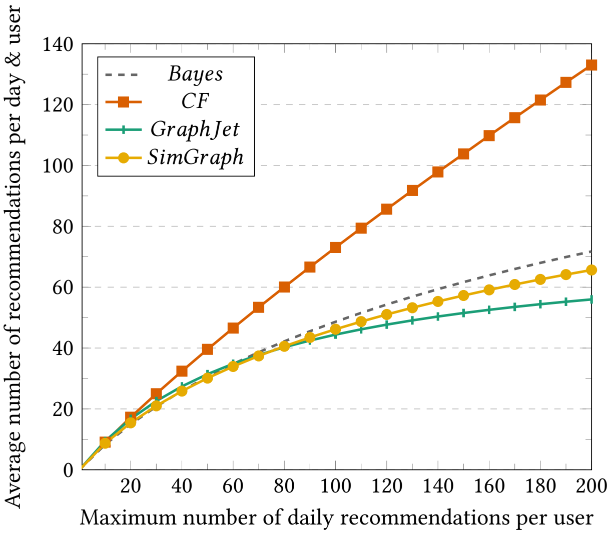

Figure 7: Recall Capacity for 1500 users

To measure the quality of the recommendations on our dataset, we compare the recommendations of the different methods with real observations. To achieve this, we consider retweets of messages which were retweeted at least twice. They constitute a set of 132, 389, 409 sharing actions for these messages that we ordered on time. We split the set in two: the first 90% of actions (the oldest retweets) compose the training set and the last 10% the test set. While the former set is used to train the four methods, the latter one allows to check the recommendations with real retweets. Note that the test set captures 66 days of retweets from the users in our dataset. Then we randomly select 500 users with less than 100 retweets (*low-active users*), 500 users with more than 100 retweets but less than 1,000 (*moderate-active users*) and finally 500 users with more than 1,000 retweets (*intensive users*). Combined, they constitute a set of 1,500 users used to compare the results of the different methods. We consider that a message is a *hit* if it is recommended to a user *before* he actually interact with the message (retweet/like). This prediction task can be seen as a quality measure.

6.2 Quality of the recommendations

Number of recommendations. Figure 7 displays the average number of recommendations proposed by the different recommender systems with respect to the maximum number of daily recommendations for each user (i.e. the size of the top-k recommendations). We observe that *Bayes*, *GraphJet* and *SimGraph* present similar behavior, capped between 50 and 70 recommendations per day and user. Only *CF*, with a linear growth, is capable of providing a high number of recommendations for any user with this maximum of 140 recommendations. There are several possible explanations for this. First, *CF* by relying on every similarity scores possible between pair of users is independent of the network. Therefore the scope of candidate items to be proposed is very large. Second, *Bayes* and *SimGraph* which rely on propagation methods could theoretically compute a prediction score for any message to almost any user. The fact that their curve is not linear seems to be a direct consequence of using thresholds during the propagation. We could argue that our threshold values are well chosen since 50 recommendations a day per user

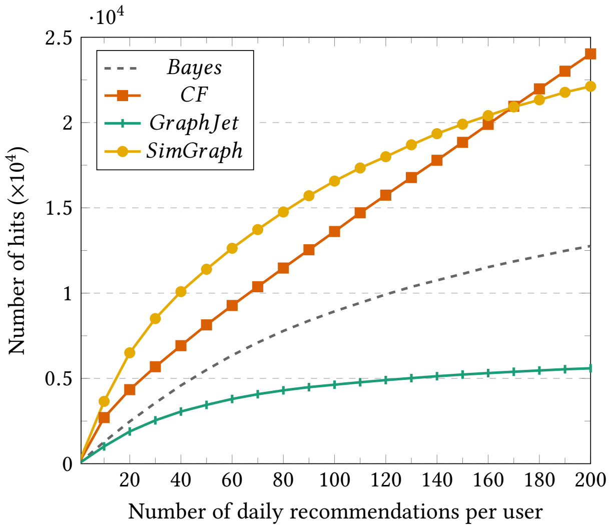

Figure 8: Number of hits for 1500 users

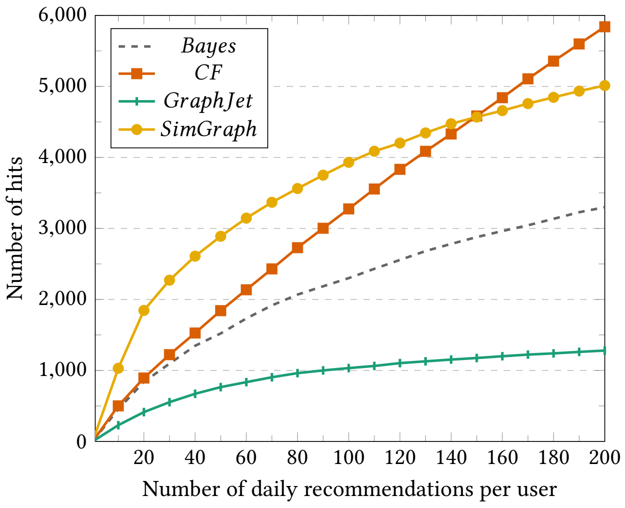

Figure 10: Number of hits for 500 *medium users*

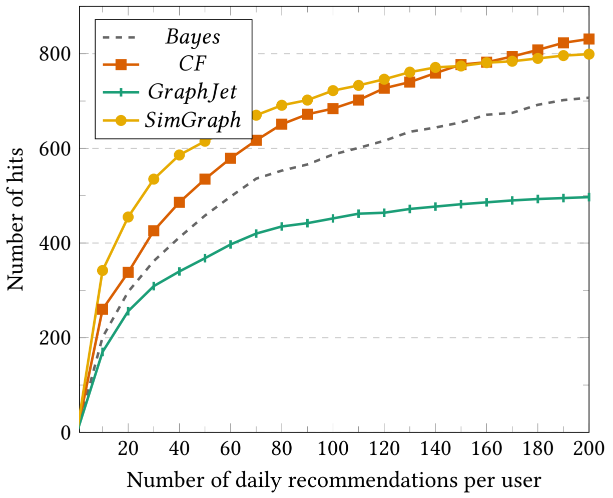

Figure 9: Number of hits for 500 *small users*

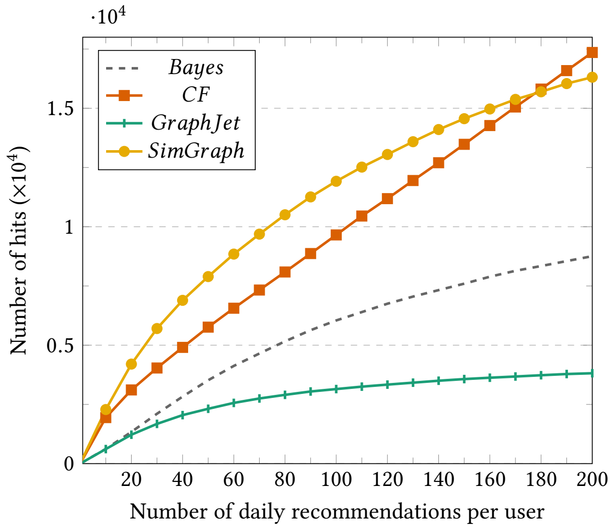

Figure 11: Number of hits for 500 *big users*

remains sufficient. Finally, *GraphJet* is limited by the low connectivity and the slow activity of small users. In fact, the total number of possibilities offered to small users is limited to their neighborhood.

Retweet prediction. A prediction corresponds to a hit in the test dataset when the recommendation of a message happens *before* it is effectively retweeted/shared in the dataset. Figure 8 plots the total number of hits for the combined set of 1,500 users with respect to the number of recommendations proposed to a user per day. This experiment is refined by different sets of users on Figure 9 (low-active users), Figure 10 (moderate-active users) and Figure 11 (big users). We observe that the results are globally stable among the different sets of considered users.

The main difference between types of users (small, medium, big) comes from the bounds of the number of hits. In fact, the total number of retweets/share is higher for big users and therefore the probability to have a hit. But interestingly, users' behavior is identical for medium and big users. According to small users, since the total number of retweets is low, the probability to find a hit grows up quickly for all approaches but a threshold is reached faster.

We note that the *CF* model has a linear evolution in the combined set of 1500 users, so it is better suited for applications proposing a high number of recommendations. But as we saw in Figure 7 this model also proposes the largest number of recommendations which leads to a constant but low precision score of 0.0001% (*i.e.*, it proposes a lot of noise). For instance, when limiting to a top-30 for recommendations, *CF* gets 5,685 hits while *Bayes* and *GraphJet* get respectively 3,564 and 2,541 hits. *SimGraph* outperforms them with a total number of 8,509 hits at top-30. However, this number of hits is limited for small users with a stabilization of 700 hits for $k > 80$. It is due to the fact that those users do not retweet much and cannot statistically lead to much more hits.

SimGraph outperforms other approaches, *e.g. GraphJet* by a factor of 3.5, for any $k < 200$. For very large values of k, *CF* slightly outperform our method *Simgraph*. Similarly to *CF*, *SimGraph* does not rely directly on the underlying network and

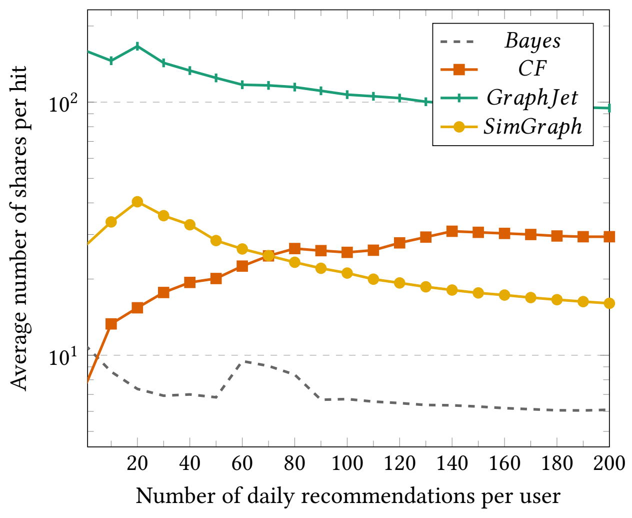

Figure 12: Popularity of hits

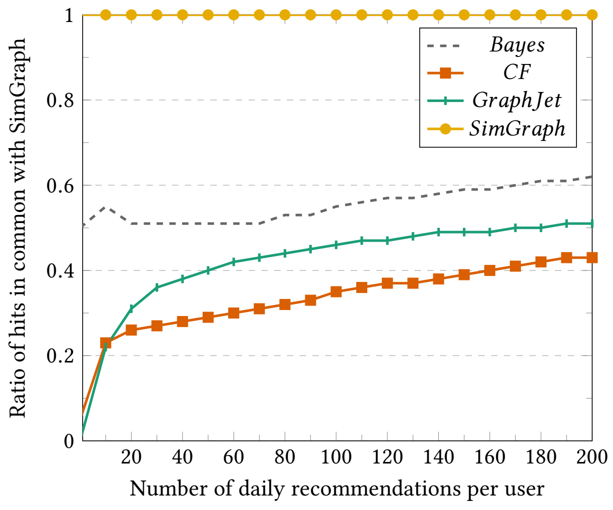

Figure 13: Parts of hits that are included in SimGraph

focuses on similarities which quickly provides good results based on users interests. But, thanks to the similarity graph and transitivity, our solution succeeds in providing new recommendations coming from other kinds of interests, different from the ones observed in the user profile, based on proximity where *CF* fails.

Popularity of the hits. We now focus on how popular the recommendations are in each solution which led to some hits. It helps to identify their scope of accuracy. Figure 12 shows that *GraphJet*'s random walks mainly generate hits on popular messages with an average number of retweets equals to 113 for each hit. Indeed, the most popular a message is, the more often the random walk will reach it. Therefore, *GraphJet* naturally is more inclined to recommend popular items.

On the contrary, the Bayesian model recommends less popular messages: for the hits it produced there were only 6 retweets on average. Thus, it produces more "local" recommendations, due to the probability computation on top of the underlying network. Observe however that the messages popularity decreases with the size of the top-k, in order to provide more recommendations *Bayes* will find items that are even less popular.

Collaborative filtering approaches, *CF* and *SimGraph*, present a more balanced result with respectively 35 and 23 retweets for a recommended tweet. These systems recommend both popular and more dedicated messages. Observe that at first *CF* will propose less popular items due to relying on strong similarities far in the network while *SimGraph* will propose more popular items. The curve will then intersect around top-70.

Hits comparison. Figure 13 displays the amount of hits in common between *SimGraph* and other methods. It shows the ratio σ of common hits, i.e. :

$$\sigma(competitor) = \frac{hits(SimGraph) \cap hits(competitor)}{hits(competitor)} \qquad (2)$$

We notice first that the ratio of common hits does not evolve more than 10% for each method. Except for *CF* that proposes less popular retweet at the beginning (see Figure 8) and more popular ones then, which levels up the number of common hits.

On the contrary, *GraphJet* focuses mainly on popularity at the beginning and shares less similarity between users, but then

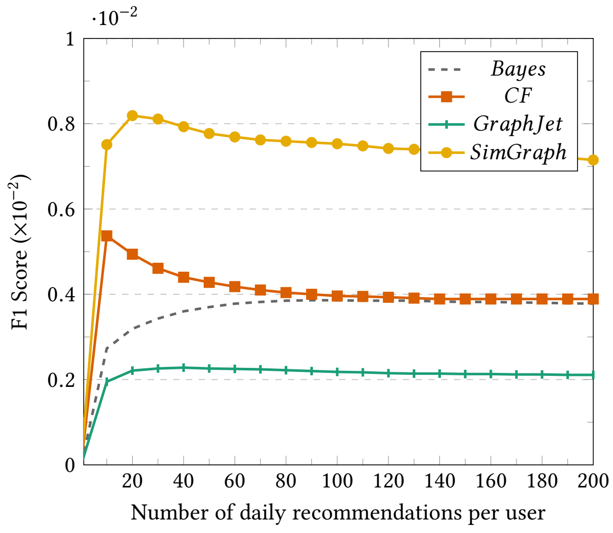

Figure 14: F1 Scores over number of recommandations

after 40 recommendations this intersection reaches a bound due to a more diversified set of recommendations in *SimGraph*.

The Bayesian method shares more similarities between hits with more than 50% which means that our method predicts retweets of tweets with various popularity: popular tweets like *GraphJet*, but also less popular tweets like *CF* and not-popular like Bayes based on the network.

F1 Measure. Figure 14 plots the F1 score for each method, according to the hits obtained over the number of daily recommendations per user. Except *Bayes*, methods peak around the same size of $k = 15$, which seems to be a good number of recommendation per day. *GraphJet* by doing less hits for same amount of recommendation leads to a lower score when combining precision and recall. *CF* proposes too many recommendations even with better precision obtains a similar F1 score as *Bayes* that proposed less recommendations but with more hits. Finally, *SimGraph* gives a really good compromise, with both popular and similar in the neighborhood, of recommendations of tweets. Overall on this task, *SimGraph* performs 4 times better than *GraphJet* and 2 times better than *Bayes* and *CF*.

6.3 Performances

Processing time. We compare now processing time for the different methods. Results are presented in Table 5. We observe that *CF* has the highest initialization time with almost 9*s* per user, since it requires to compute the similarities between all pairs of users, so theoretically $|V|^2 \approx 1.3\times10^{12}$ computations. Oppositely *Bayes* has the lowest initialization time with 10*ms* per user to initialize all probabilities. For *SimGraph*, the initialization consists in exploring for each user his neighborhood up to distance 2 (BFS exploration) and to compute similarities for each user found during the exploration to build the similarity graph. For each user, using this method requires 311*ms* to determine its most similar users. *GraphJet* only relies on the Twitter and retweets graph and does not need any initialization.

According to the message processing time, all the methods were implemented in a multi-thread way on a 70 cores parallelization process. We can observe that Bayes takes around 1 second to compute the recommendation scores for a given message due to graph exploration. Oppositely *CF* is able to compute very quickly the different recommendations (0.5*ms* per message) based on the pre-computed similarities. *SimGraph* requires 38*ms* to compute the recommendation score of a message to users based on our iterative propagation algorithm. *GraphJet* is user-centric and computes top-k recommendations for a given user, and not a set of users to recommend a given message. Computing k recommendations for a given user requires 14*ms*.

To compare performances of the different algorithms we measure the total time for processing both the initialization and the recommendations, for each user for *GraphJet* or for each of the 13.2 million incoming messages during the test dataset for other methods. For *GraphJet* we perform the computation periodically, here we chose every 5 hours. Note that the test period extends over 66 days. It results that *Bayes* is the most expensive method with 51.22*h* of total computation. *CF* is also expensive with 41.01*h* of computation mainly for the pre-processing. *GraphJet* provides much faster recommendations (4.2*h*). *SimGraph* is a good trade-off with an initializing time by node of 311*ms* and an average time per message of 38*ms* leading to a total computation time of 3.41*h*. We see that both the *Bayes* and *CF* methods suffer of very long computation time, making them hard to use in a real life scenario.

Notification time. This experiment illustrates the benefit of the different methods regarding the notification time. In other words, we want to study how much time in advance a recommended tweet will be presented to the user before the actual interaction. Remember that according to our study in Section 3, a tweet has a short lifespan. We compute for each hit the difference between the time when the recommendation was performed and the time when the user perform the retweet in the test dataset. In Figure 15, we see that *GraphJet* is very stable and permits to predict a hit 80,000s (around 22 hours) in advance on average. *GraphJet*'s tendency to recommend popular items makes it more efficient on this task, having more opportunities to predict such messages before. Interestingly the*CF* curve is correlated with the average popularity of the item predicted in Figure 12. Recommending dedicated items long time in advance is a difficult task therefore, and naturally the average advance time increases for *CF* and

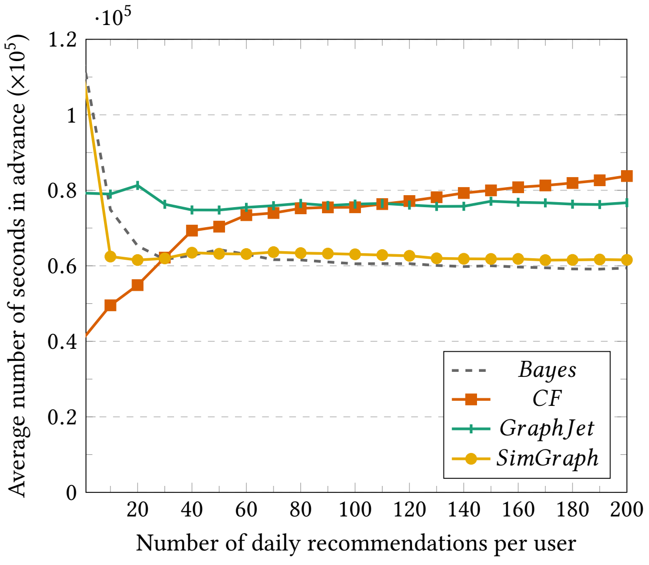

Figure 15: Average advance time before real retweets (in s)

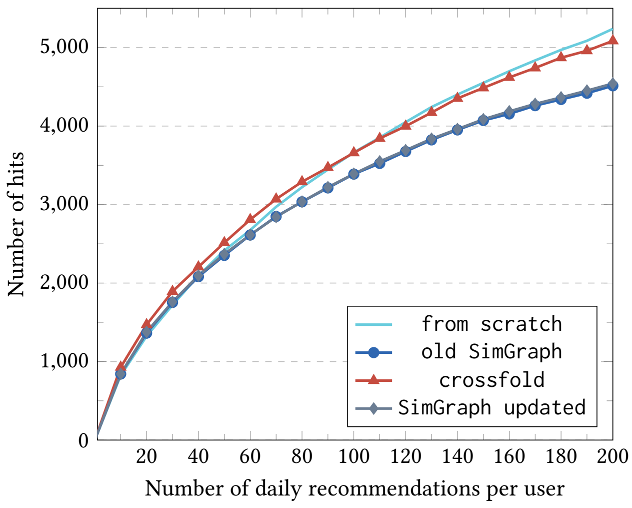

Figure 16: Number of hits with several updating strategies

can predict recommendations before *GraphJet* for very large k values. Indeed *CF* benefits from not having to wait for a message to propagate through the network to compute a recommendation score, therefore can predict popular items long time in advance. Following the same conclusion that predicting dedicated messages long time in advance is a difficult task, it fully explain why *Bayes* and *SimGraph* can only predict iterations 17h in advance on average since they have to wait for more signals.

Graph update. Microblogging platforms such as Twitter are permanently in motion. Each retweet/like performed impacts some changes of similarity scores and consequently impacting recommendations computation and quality. Keeping structures up to date to fit the shift of users' interest is a very difficult task. Therefore an efficient updating strategy is crucial to make our recommendation model robust to network and retweet evolutions. *GraphJet* is a very suitable solution in real life because, by avoiding any initialization step, it continuously stays updated no matter the amount of new information published. However, as

	init. (per user)	init total time 1,149,374 users	time (per message)	total time (70 cores //) 13,238,941 Tweets (Trial period)	total time init + recos
Bayes	10ms	0.04h	975ms	51.22h	51.26h
CF	8,583ms	39.40h	0.5ms	0.02h	41.01h
SimGraph	311ms	1.41h	38ms	2.00h	3.41h
	init. (per user)	**init total time** 1,149,374 users	**time (per user)**	**total time (70 cores //)** 1,149,374 users * 66 days (Trial period)	**total time** init + recos
GraphJet	0ms	0h	14ms	4.2h	4.2h

Table 5: Initialization and recommendation time (in ms)

we saw in Figure 8 the gain in hits prediction with our method *SimGraph* is substantial enough to inquire how we can keep this increase of accuracy while dealing with incremental updates. We study the impact of computing our recommendations based on an outdated similarity graph and the outcome of using different updating strategies. For this experiment we assume that *SimGraph* was built at the beginning of the test dataset (after 90% of the retweets). We plot in Figure 16 the number of hits obtained for the last 5% of the retweets (half of the 10% trial). We compare four approaches:

- `from scratch`, where the graph is totally rebuilt after 95% of the retweets, from the original graph,
- `old simgraph`, where we keep the exact same *SimGraph* that the one computed at 90%,
- `crossfold`, where we apply our similarity graph construction on the previous `old simgraph` instead of the twitter graph,
- `SimGraph update`, where we update similarity scores on the similarity graph built at 90%

As expected re-building `from scratch` the similarity graph at 95% allows to get the best predictions. This strategy is also the most expensive one with similar computation cost as the ones expressed in Table 5. Surprisingly `old SimGraph` and `SimGraph update` have almost the exact same results, indicating that the topology of the computed network has a more important impact than the weight computed on the edges. The `crossfold` strategy is very promising because it fits almost perfectly with the `from scratch` strategy while cutting drastically computation cost. Indeed, the `crossfold` strategy perform a BFS at distance 2 on the already computed similarity graph to search for new influential users that weren't considered during the first *SimGraph* computation. This method both increases the density of the graph while updating the weight edges. These results enlighten the possibility to follow the evolution of users by incrementally computing a *SimGraph* on top of the previous iteration and avoid computing things from scratch.

7 CONCLUSION

We propose in this paper *SimGraph*, a scalable recommendation model based on a similarity graph which exploits homophily for propagation of probabilities. We study the homophily impact between users on the network. This indicate that homophily is a good property to find quickly users of high similarities and therefore drastically reduce the computation cost of such operation. We find that applying transitivity on those similarities reduces the sparsity efficiently with a fast convergent model thanks to optimization technics. We propose a propagation model in order to compute on-demand relevant recommendations for an incoming post. Our experiments enlighten that our method outperforms other approaches for recommendation computations, but also provides more hits than the competitors for a lower number of recommendations blending popular and more confidential items. We also demonstrate how *SimGraph* can be updated at low cost, showing its usability in real world scenarios.

As future works, we intend to enhance our similarity graph by analyzing content of the tweets with entity recognition. In fact, our similarity is based on common retweets between users and can be improved by creating "topic tweets" by merging similar tweets. This will make users likely to be similar in the similarity graph and therefore enhance results for small users. We also plan to break "information bubbles", since recommended information is generally originated from the same sub-part of the graph. We are currently working on the identification of bubbles in our twitter graph based on both the network topology and tweet topics. Then we will propose a complementary score for recommendations by escaping from information locality from a bubble to another.

REFERENCES

[1] Gediminas Adomavicius and Alexander Tuzhilin. 2005. Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-art and Possible Extensions. *Trans. on Knowledge and Data Engineering (TKDE)* 17, 6 (2005), 734–749.

[2] Hyung Jun Ahn. 2008. A New Similarity Measure for Collaborative Filtering to Alleviate the New User Cold-starting Problem. *Inf. Sciences* 178, 1 (2008), 37–51.

[3] Parantapa Bhattacharya, Muhammad Bilal Zafar, Niloy Ganguly, Saptarshi Ghosh, and Krishna P. Gummadi. 2014. Inferring User Interests in the Twitter Social Network. In *Proc. Intl. Conf. on Recommender Systems (RECSYS)*. 357–360.

[4] John S. Breese, David Heckerman, and Carl Kadie. 1998. Empirical Analysis of Predictive Algorithms for Collaborative Filtering. In *Proc. Intl. Conf. on Uncertainty in Artificial Intelligence (UAI)*. 43–52.

[5] Elanor Colleoni, Alessandro Rozza, and Adam Arvidsson. 2014. Echo Chamber or Public Sphere? Predicting Political Orientation and Measuring Political Homophily in Twitter Using Big Data. *Jour. of Communication* 64, 2 (2014), 317–332.

[6] Camelia Constantin, Ryadh Dahimene, Quentin Grossetti, and Cédric Du Mouza. 2016. Finding Users of Interest in Micro-blogging Systems. In *Proc. Intl. Conf. on Extending Database Technology (EDBT)*. 5–16.

[7] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep Neural Networks for YouTube Recommendations. In *Proc. Intl. Conf. on Recommender Systems (RECSYS)*. 191–198.

[8] Peter Forbes and Mu Zhu. 2011. Content-boosted Matrix Factorization for Recommender Systems: Experiments with Recipe Recommendation. In *Proc. Intl. Conf. on Recommender Systems (RECSYS)*. 261–264.

[9] Blaž Fortuna, Carolina Fortuna, and Dunja Mladenić. 2010. Real-time News Recommender System. In *Proc. Eur. Conf. on Machine Learning and Knowledge Discovery in Databases (ECML-PKDD)*. 583–586.

[10] Fréderic Godin, Viktor Slavkovikj, Wesley De Neve, Benjamin Schrauwen, and Rik Van de Walle. 2013. Using Topic Models for Twitter Hashtag Recommendation. In *Proc. Intl. World Wide Web Conference (WWW)*. 593–596.

[11] David Goldberg, David Nichols, Brian M. Oki, and Douglas Terry. 1992. Using Collaborative Filtering to Weave an Information Tapestry. *Commun. ACM* 35, 12 (1992), 61–70.

[12] Pankaj Gupta, Ashish Goel, Jimmy Lin, Aneesh Sharma, Dong Wang, and Reza Zadeh. 2013. WTF: The Who to Follow Service at Twitter. In *Proc. Intl. World Wide Web Conference (WWW)*. 505–514.

[13] Jonathan L Herlocker, Joseph A Konstan, Al Borchers, and John Riedl. 1999. An algorithmic framework for performing collaborative filtering. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 230–237.

[14] Zan Huang, Hsinchun Chen, and Daniel Zeng. 2004. Applying Associative Retrieval Techniques to Alleviate the Sparsity Problem in Collaborative Filtering. *ACM Trans. Inf. Syst.* 22, 1 (Jan. 2004), 116–142.

[15] Jeon hyung Kang and Kristina Lerman. 2012. Using Lists to Measure Homophily on Twitter. In *AAAI work. on Intelligent Techniques for Web Personalization and Recommendation*.

[16] Mohsen Jamali and Martin Ester. 2010. A matrix factorization technique with trust propagation for recommendation in social networks. In *Proceedings of the fourth ACM conference on Recommender systems*. ACM, 135–142.

[17] Meng Jiang, Peng Cui, Rui Liu, Qiang Yang, Fei Wang, Wenwu Zhu, and Shiqiang Yang. 2012. Social Contextual Recommendation. In *Proc. Intl. Conf. on Information and Knowledge Management (CIKM)*. 45–54.

[18] Danae Pla Karidi, Yannis Stavrakas, and Yannis Vassiliou. 2017. Tweet and Followee Personalized Recommendations Based on Knowledge Graphs. *Jour. of Ambient Intelligence and Humanized Computing* (2017), 1–15.

[19] Yehuda Koren. 2008. Factorization Meets the Neighborhood: a Multifaceted Collaborative Filtering Model. In *Proc. Intl. Conf. on Knowledge Discovery and Data Mining (KDD)*. 426–434.

[20] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix Factorization Techniques for Recommender Systems. *Computer* 42, 8, 30–37.

[21] Haewoon Kwak, Changhyun Lee, Hosung Park, and Sue Moon. 2010. What is Twitter, a Social Network or a News Media?. In *Proc. Intl. World Wide Web Conference (WWW)*. 591–600.

[22] Kristina Lerman, Rumi Ghosh, and Tawan Surachawala. 2012. Social Contagion: An Empirical Study of Information Spread on Digg and Twitter Follower Graphs. *CoRR* abs/1202.3162 (2012).

[23] Pasquale Lops, Marco De Gemmis, and Giovanni Semeraro. 2011. Content-based Recommender Systems: State of the Art and Trends. In *Recommender Systems Handbook*. Springer, 73–105.

[24] Hao Ma, Haixuan Yang, Michael R Lyu, and Irwin King. 2008. Sorec: Social Recommendation Using Probabilistic Matrix Factorization. In *Proc. Intl. Conf. on Information and Knowledge Management (CIKM)*. 931–940.

[25] Paolo Massa and Paolo Avesani. 2007. Trust-aware Recommender Systems. In *Proc. Intl. Conf. on Recommender Systems (RECSYS)*. 17–24.

[26] Arthur Mensch, Julien Mairal, Bertrand Thirion, and Gaël Varoquaux. 2016. Dictionary Learning for Massive Matrix Factorization. In *Proc. Intl. Conf. on Machine Learning (ICML) (ICML'16)*. 1737–1746.

[27] Bradley N. Miller, Istvan Albert, Shyong K. Lam, Joseph A. Konstan, and John Riedl. 2003. MovieLens Unplugged: Experiences with an Occasionally Connected Recommender System. In *Proc. Intl. Conf. on Intelligent User Interfaces (IUI)*. 263–266.

[28] N.Koumchatzky and A.Andryeyev. 2017. Using Deep Learning at Scale in Twitter's Timelines. (2017). https://blog.twitter.com/2017/using-deep-learning-at-scale-in-twitter-s-timelines

[29] Aäron van den Oord, Sander Dieleman, and Benjamin Schrauwen. 2013. Deep Content-based Music Recommendation. In *Proc. Intl. Conf. on Neural Information Processing Systems (NIPS)*. 2643–2651.

[30] Alan Said and Alejandro Bellogín. 2014. Comparative Recommender System Evaluation: Benchmarking Recommendation Frameworks. In *Proc. Intl. Conf. on Recommender Systems (RECSYS)*. 129–136.

[31] Sebastian Schnettler. 2009. A Structured Overview of 50 Years of Small-World Research. *Social Networks* 31, 3 (2009), 165 – 178.

[32] Aneesh Sharma, Jerry Jiang, Praveen Bommannavar, Brian Larson, and Jimmy Lin. 2016. GraphJet: Real-time Content Recommendations at Twitter. *Proc. VLDB Endow.* 9, 13 (2016), 1281–1292.

[33] Jiliang Tang, Xia Hu, and Huan Liu. 2013. Social recommendation: a review. *Social Network Analysis and Mining* 3, 4 (2013), 1113–1133.

[34] Ibrahim Uysal and W Bruce Croft. 2011. User Oriented Tweet Ranking: a Filtering Approach to Microblogs. In *Proc. Intl. Conf. on Information and Knowledge Management (CIKM)*. 2261–2264.

[35] Jianshu Weng, Ee-Peng Lim, Jing Jiang, and Qi He. 2010. TwitterRank: Finding Topic-sensitive Influential Twitterers. In *Proc. Intl. Conf. on Web Search and Data Mining (WSDM)*. 261–270.

[36] Xiwang Yang, Yang Guo, and Yong Liu. 2013. Bayesian-Inference-Based Recommendation in Online Social Networks. *IEEE Trans. Parallel Distrib. Syst.* 24, 4 (2013), 642–651.

[37] Faiyaz Al Zamal, Wendy Liu, and Derek Ruths. 2012. Homophily and Latent Attribute Inference: Inferring Latent Attributes of Twitter Users from Neighbors.. In *ICWSM*.