# Graph-based recommendations for big data systems

Lev Denisov, Ferdiansyah Dolot, Anna Turu Pi

*Universitat Politècnica de Catalunya*

June 25, 2019

**ABSTRACT** Recommendation systems use the data available about users' preferences towards a given set of items to compute predictions about the unknown preferences. Over the last decades, the web and the data management technologies have been evolving exponentially increasing the user data available. However, we detect that the common practice in recommendation systems hasn't evolved and they still use matrix-based algorithms, which don't support Big Data Systems full potential. Twitter and Pinterest have already developed graph-based systems to substitute the traditional methods to obtain more reliable results. The purpose of this paper is to study the state of the art in graph-based recommendation systems, and assess if the systems developed support real-time recommendations in large-scale datasets.

## Introduction

Many current approaches in research in the area of recommendation systems though giving good results on benchmarks are often impractical to implement in the real production environments. It was shown in Netflix Prize competition [1] where the solution was evaluated as "the additional accuracy gains that we measured did not seem to justify the engineering effort needed to bring them [the winning methods] into a production environment". Many companies now realize that they need recommendations that work with real time data as it gives much better user engagement. Most of the traditional recommendation algorithms do not scale well and are unable to work with real time data. So companies start using graph-based recommendation algorithms that look very promising in terms of scalability. Based on this we decided to explore state-of-the-art in graph-based recommendation algorithms deployed in production environments in different companies.

Recommendation systems collect information on the preferences of its users for a set of items. [2] Its objective is to predict the preferences of the users in items they haven't explicitly stated their preference yet (by ratings). The field of recommendation systems has been evolving as more information was made available. During the time of the static Web, recommendation systems were very limited, using as a main source of data demographic information. After the Web 2.0, recommendation systems also include social information (followers, likes), information that the users include dynamically. Currently, massive datasets of IoT data can be included in the predictions (GPS location, RFID). Consequently, first recommendation systems were static and supported a limited amount of data. Most of the companies still use traditional recommendation systems because of the challenges of supporting scalability and real-time data.

In the next sections we present how traditional recommendation systems work. After explaining its limitations, we focus on graph recommenders and how they tackle these challenges.

## 1 Traditional recommendation systems

As shown in Figure 1, a recommendation system is composed of the following structure: [2]

The data from the different data sources is used in a filtering algorithm: it can be demographic, content-based, collaborative, or an hybrid filtering by merging more than one technique. Then it creates a model. It is created using a "memory-based" method if it uses the data directly, without processing, or it uses a "model-based" method if the model is generated from the data. From that model, it predicts the preference level of each user towards each item. From those, each user is recommended the subset of items with a higher predicted preference. The model employed can be validated by applying different metrics on the results.

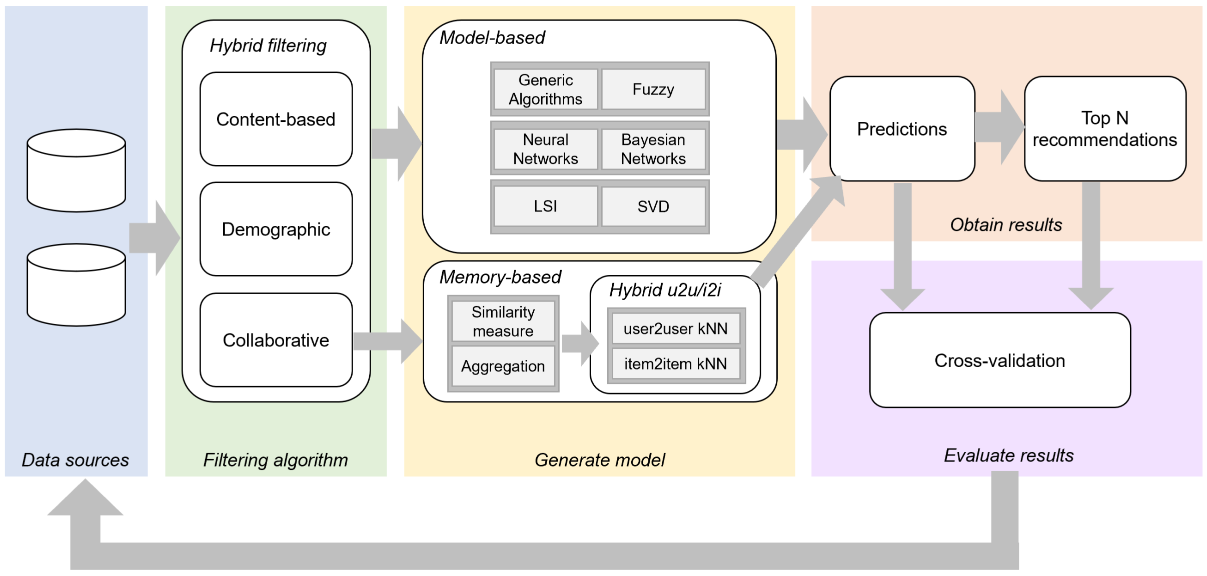

Figure 1: Traditional models of recommendations and their relationships, extracted from J.Bobadilla, *Recommendation Systems survey* [2]

## 1.1 Content-based filtering

Content-based filtering is based only on the user choices made in the past. [2] For that, the system compares attributes of items for which the user has shown some preference (by rating or purchasing) with items with no information towards the preference.

The methodology used is to compute the similarity between items. Similarity is a metric that aggregates the number of properties two items have in common. When generating content-based filtering predictions for a given user, the recommender will suggest to the user the items that have the highest similarity with the items the user has a high interest in.

The drawbacks of using content-based filtering are the limited content analysis and the overspecialization. To compute more reliable predictions it is more convenient to have a wider view of the system and not to base the model only in the information of a given user. Hence collaborative filtering is the most widely used technique in recommenders.

## 1.2 Collaborative filtering

To predict the preference level of each item for a given user, apart of using the given user's own experience on other items, it uses the experience from their group of acquaintances. Hence it provides richer predictions than content-based filtering, as it takes into account that users might be interested in items that people from their close environment are interested in too.

### 1.2.1 User-to-user kNN algorithm

The technique most commonly used in recommenders is the user-to-user k-NN algorithm. It comprises the 3 following steps [2]:

1. Determine k user nearest neighbors (i.e. neighborhood) for user $a$
2. Aggregate the ratings of the neighborhood about items not rated by $a$
3. Extract predictions, and from those, the top N recommendations

To show how the algorithm works, we reuse the example from J.Bobadilla, *Recommendation Systems survey* [2]. Here we apply a user-to-user kNN algorithm, being k = 3, the similarity measure is computed as 1 – (mean squared differences), and the aggregation metric used is the average.

To determine the k nearest neighbors for the user 4, first the similarity between the given user and every other user is computed. If the users have no item in common, as the case of sim(4,1), they have no similarity. The neighborhood of the user 4 is composed of the users with a highest similarity, in this case users 2, 5 and 7.

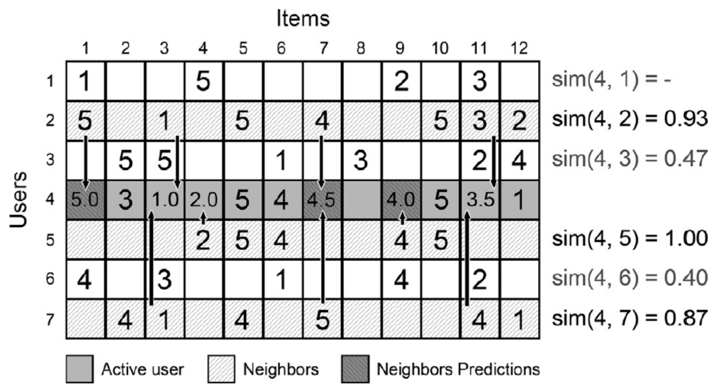

Figure 2: User-to-user kNN algorithm example extracted from J.Bobadilla, *Recommendation Systems survey* [2]

### 1.2.2 Item-to-item kNN algorithm

This method considers finding users similar to each other, as users interested in a given user might also be interested in similar items. It is composed of those 3 steps [2]:

1. Determine $q$ neighbors for each item in the system
2. Given a user $a$, for each item $i$ where the user's preference is unknown, calculate its prediction based on the ratings of $a$ from the $q$ neighbors of $i$
3. Select the top N recommendations for the user $a$, i.e. the N items with the highest predicted value

This approach has some advantages compared to the user-to-user version. The similarity measure needs to be computed every time new possible neighbors are registered in the database. The expected growth in items is less steep than in users, reducing the scalability problem significantly. Also, outdated information for items is less sensitive than for the users. [2]

### 1.2.3 Pros and cons of Collaborative Filtering based on kNN

K Nearest Neighbor is the most widely spread algorithm for recommenders as it is conceptually simple. Its straightforward implementation tips the scales in its favor against alternatives. In result, its implementations have obtained good quality predictions and recommendations.

On the contrary, in datasets with a high level of sparsity, the similarity measures are hard to compute due to insufficient values in common, leading to less precise predictions. kNN algorithm is also unreliable in cold start situations, i.e. where data is scarce. Moreover, a determining factor that makes kNN collaborative filtering unfeasible for Big Data systems is its low scalability. [2]

## 1.3 Challenges

### 1.3.1 Scalability and sparsity problem

In the beginning of traditional recommendation systems the data available was limited. In the recent years, this amount has increased exponentially, generating a scalability problem that requires different approaches. The number of items contained in a recommendation system is enormous, and only a few are popular among the users. The figure 3 plots in the x-axis the number of items available in a system, and the y-axis shows the popularity of each item. It represents the fact that physical organizations can only provide the most popular items, while on-line organizations can include everything available. [3] This Long Tail shows that there is a sparsity problem. The corresponding utility matrix is sparse if for each user the system only contains information for a small set of items. Having a sparse utility matrix complicates the predictions.

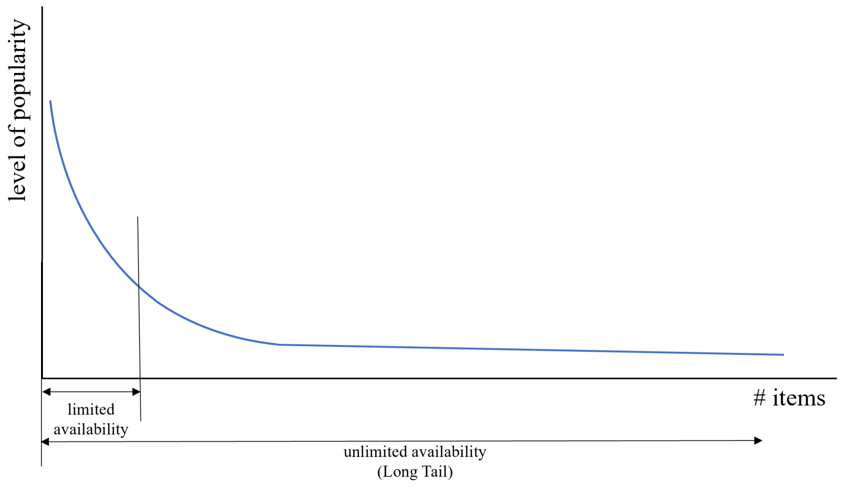

Figure 3: Long tail

### 1.3.2 Cold-start problem

The initial lack of information generates unreliable recommendations. [2] There are 3 kinds of cold-start:

**New community:** when starting a recommendation system.

**New item:** an item that is not rated will not be recommended, hence it goes unnoticed.

**New user:** doesn't receive personalized recommendations. When the user provides the first ratings, the system bases its predictions in a small amount of evidence, hence the predictions are imprecise and the user stops using the service.

In the next section we list different alternatives to reduce the scalability, sparsity and cold-start problems.

## 1.4 Data mining technologies in recommendation systems

*Association rules*
The cold-start problem can be reduced using cross-level association rules about items' domain. [2] Association rules are used to find association between two sets of items so that the occurrence of items in one set implies the presence of the items from the other set. [4]

*Collaborative tagging*
An effective approach used to solve data sparseness and cold-start users is **collaborative tagging to filter users' interest**. A common example is to crawl the collaborative tagging from Delicious. [2] [5] [6]

*Bayesian networks*
An alternative to the utility matrix to reduce the sparsity and scalability problem is to use **classification algorithms**, i.e. to generate a decision tree for every user to decide if an item is of their liking. [3] Bayesian networks create a model based on a training set with a decision tree at each node and edges representing user information. The model can be built off-line over a matter of hours or days. The resulting model is very small, very fast, and essentially as accurate as nearest neighbor methods. [4] This solution is unfeasible for Big Data, as it takes a long time to construct and it would be expensive to update.

*Dimensionality reduction*
An alternative approach to estimate blank entries in an utility matrix is to use **dimensionality reduction**. Dimensionality reduction can either be based on matrix factorization or a combination of latent semantic index (LSI) and singular value decomposition (SVD). [3] [2] However, SVD is expensive and would only be used in static settings, where the known information doesn't change in time. Also, LSI can reduce the sparsity to a certain extent, but it will lose some information and can't reduce the huge number of users that is the most important reason of reducing the system scalability. [4]

*Clustering techniques*
An alternative strategy to improve the prediction quality and reduce scalability, sparsity and the cold-start problem is to cluster users or items into groups. [3] [2] [4] Clustering techniques work by identifying groups of users who appear to have similar preferences. Once the clusters are created, predictions for an individual can be made by averaging the opinions of the other users in that cluster. [4] First, the users are grouped in clusters based on the similarity between each other. Next, detect the k users' clusters nearest neighbors for each user *U*. Lastly, the top N recommendations are derived from the list of k users' clusters (refer to Figure 4).

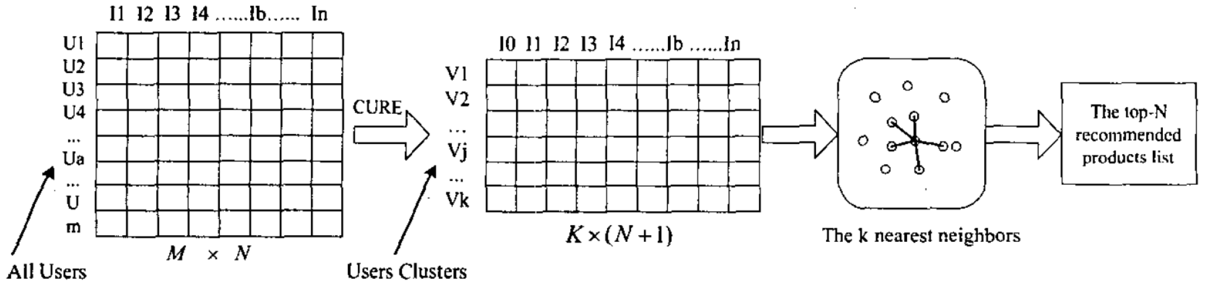
I1 I2 I3 I4 ......Ib...... In
U1
U2
U3
U4
...
Ua
...
U m
All Users
M × N
CURE
I0 I1 I2 I3 I4 ......Ib ......In
V1
V2
...
Vj
...
Vk
Users Clusters
K×(N+1)
The k nearest neighbors
The top-N recommended products list

Figure 4: Improved Collaborative Filtering method, extracted from W.Yang (2004) [4]

### 1.4.1 Change of paradigm in recommendation systems

Those methods manage to reduce the recommenders' problems to some extent, but they are still in line with the implementation of collaborative filtering kNN algorithm. The next section describes a completely different methodology to create recommendation systems, based on graphs. With graph-based recommendation systems scalability problem has a few viable approaches. Thus they provide a feasible solution for Big Data systems. What's more, they are easier to build than models based on filtering, easier to maintain and adapt to changes.

## 2 Graph-based recommendation systems

Using graphs is the approach that has managed to solve more effectively the sparsity and scalability problem in Recommendation Systems so far. Apart from that, research shows that it is possible to use graphs and Linked Open Data [7] to complement insufficient features which may be important for content-based recommenders in some cases. In addition, contrary to collaborative filtering, graphs include connections from social network that add logics to recommendations. [8]

We are interested in particular in the graph-based recommendations approaches that can be applied in systems with large number of items. There are number of approaches for graph-based recommendations systems that are actually applied in large production systems. The most popular approach is to use random walks as it is a well scaling approximate algorithm to obtain ranking of the nodes. Also there are few other approaches that are not as popular but nonetheless used in companies: Online Motif Detection and Graph Convolutional Networks (GCN). All these algorithms are used as a part of large-scale recommendation systems. In this section we will review these methods.

## 2.1 Horting algorithm

B.Mirza et al. proposed a first graph-based technique to reduce the sparsity in user-based Recommendation Systems [9], based on the Horting algorithm [10]. Horting graph is a graph-based technique in which nodes are users, and edges between nodes represent the degree of similarity between two users. [4] Recomendation systems using the Horting algorithm provide better predictions than nearest neighbor algorithms [10]. While nearest neighbor algorithms would be the equivalent to adjacency operations, the Horting algorithm allows to traverse the graph: predictions are produced by walking the graph to nearby nodes and combining the opinions of the nearby users. [4]

### 2.1.1 Skip jump

The example displayed in Figure 5 shows how a recommendation system can be build by bringing different graphs (representing different sources) together. The skip jump, by matching people nodes from graph *a* to the corresponding nodes in graph *b*, generates a graph that includes the connections from both sources. This is a preliminary step to understand the next section.

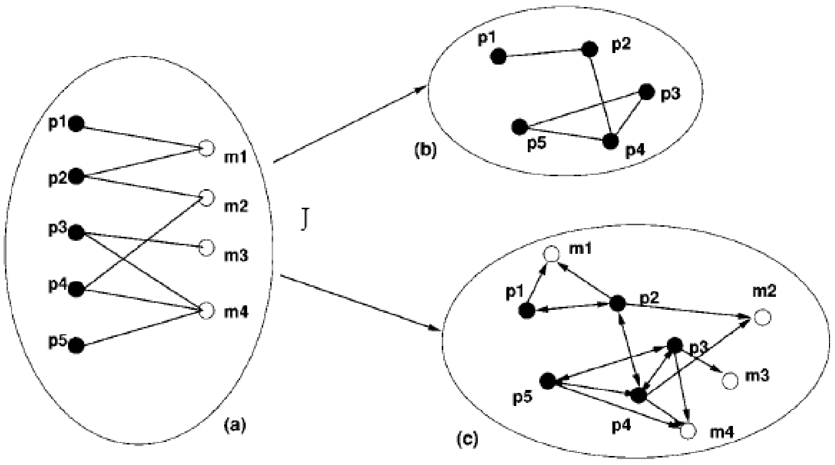

Figure 5: Skip jump. (a) Bipartite graph of people and movies. (b) Social network graph, and (c) recommender graph. Extracted from B. J. Mirza (2003)

### 2.1.2 The hammock jump

The hammock jump brings 2 people together if they have at least $w$ movies in common, being $w$ the hammock's width. The Horting algorithm consists in a sequence of hammock jumps. [8] The Figure 6 shows a path of hammock jumps, with a common width of w = 4 (example extracted from B. J. Mirza (2003) [8]). On the right we display the relations between nodes inferred by the Horting algorithm.

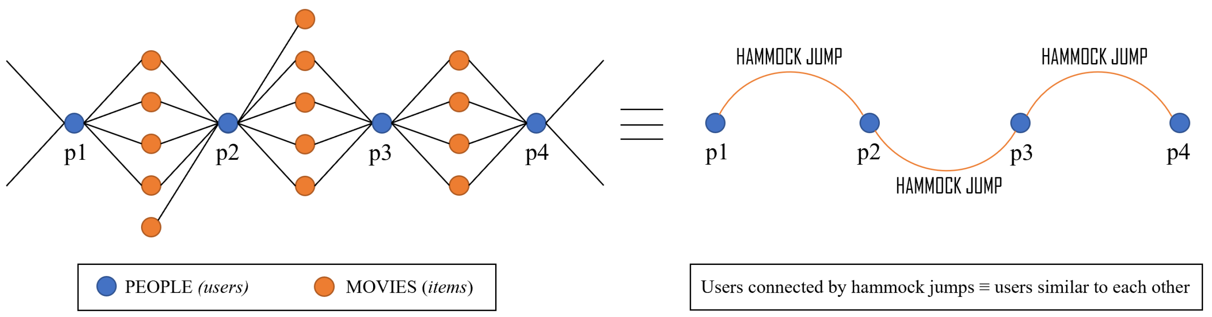

Figure 6: Left: a path of hammock jumps, extracted from B. J. Mirza (2003). Right: our interpretation

## 2.2 Random walk based algorithms

Most of the production graph-based recommendation systems use algorithms that are variations and extensions of Random Walk on graph going back to [11]. Random walk - stochastic algorithm that allows sampling neighbor nodes and exploring neighbors of neighbors. This algorithm usually ranks nodes by the number of visits. The main advantage of the algorithm is that it can be applied on the arbitrarily large graphs in time that does not depend on the size of the graph. The algorithm gives an approximate rating with the precision depending on the number of steps. Another advantage is its simplicity. In table 1 we summarize algorithms based on random walk or extending on it.

Table 1: Algorithms based on Random Walk

| **Algorithm** | **Products** | **Description** | **Purpose** |
|---|---|---|---|
| Biasing random walk with user features [12] | Pixie | When selecting neighbor nodes have higher probability of selecting nodes with similar to user features (e.g. language). | More personalized recommendations |
| Start random walk from multiple query items with weights [12] | Pixie | Have more than one starting point for random walks and intersect results obtained from different starting points. Assign number of steps to the starting point proportional to its degree (non-linear). | More relevant recommendations |
| Multi-hit booster [12] | Pixie | Boost weight for the items reached from random walks started from different items. If an item is reached from more than one starting point assign higher rank to it. | More relevant recommendations |
| Early stopping of random walk [12] | Pixie | Use heuristic: stop random walks if at least $N_p$ candidates were visited at least $N_v$ times. | Faster execution |
| Egocentric random walk (personalized PageRank [11, 13]) | WTF, RealGraph, GraphJet | Random walk with probability to jump back to the starting set. This does not allow the algorithm to stray too far. | More relevant recommendations |
| SALSA [13, 14] | WTF, RealGraph, GraphJet | Random walk in a bipartite graph with ”hubs” on one side and ”authorities” on another. Each step traverses two edges of bipartite graph - one forward and one backward and assigns scores to both sides. | More relevant recommendations, more diverse recommendations |
| Subgraph SALSA [13] | GraphJet | Materializes a subset of graph (for example from egocentric random walk) in memory and runs SALSA on this subgraph distributing weights according to the node degree. | More relevant recommendation, decrease diversity of recommendations to provide reason why item got recommended, small memory footprint, faster computations |
| Cosine Follow [13, 15] | GraphJet | Random walk assigning rank to the nodes approximating the cosine similarity between the source node and its neigbors. Used to find similar nodes of the same type. | Scalable finding of similar items |

## 2.3 Online motif detection

Another approach to recommendations in graphs detection of particular meaningful structures in the graph – motifs. Network motifs are recurrent patterns in graphs, usually defined in terms of vertices and edges that are arranged in a specific configuration. Nearly all approaches to motif detection are based on a static graph snapshot and viewed as batch computations. The online version of this approach is possible: to identify motifs as they are being formed in real time and trigger appropriate actions.

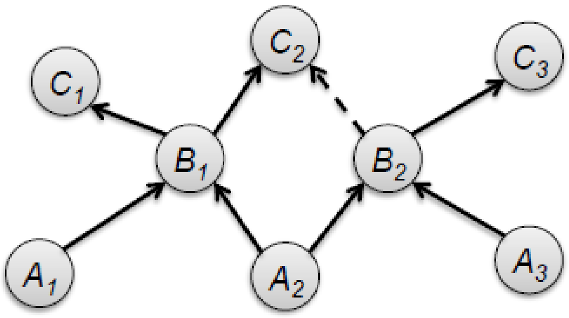

Figure 7: Sample fragment of the follow graph: when the edge $B_2 \rightarrow C_2$ is created, we want to push $C_2$ to $A_2$ as a recommendation. Image extracted from [13]

In terms of notation, $A$'s refer to users for whom we wish to make recommendations; $B$'s are the users whom the $A$'s follow, and $C$'s are the users whom the $B$'s follow. We want to recommend the appropriate $C$'s to the $A$'s. This shows the basic case and in general any node can be in any role $A$, $B$ or $C$. For simplicity, let us assume k = 2 in the above definition, which means that when the edge $B_2 \rightarrow C_2$ is created in Figure 7, we want to push $C_2$ to $A_2$ as a recommendation. In other words, we are interested in the "diamond" motif involving four graph vertices. So far in the literature the only system that uses this approach is MagicRecs [16]. They discovered that the problem of detection of such motifs on-line can be reformulated and optimized in terms of intersection of adjacency lists. In the case of MagicRecs the graph itself is still static and recomputed every n hours. The algorithm only reacts to the cases when the new edge completes the pattern in the graph.

## 2.4 Graph Pruning

Graph Pruning is an auxiliary algorithm that helps to reduce size of the graph and improve the recommendations quality by removing irrelevant edges. This algorithm was proposed in [12] for Pinterest recommendation system called Pixie. The algorithm was designed to work with a bipartite graph consisting of pins and boards. Where pins are visual bookmarks containing a description, a link, and an image or a video. Users at Pinterest view pins and curate them into collections called boards. The graph pruning algorithm prepares graph for running recommendations algorithms by deleting edges that connect pins to boards that are not thematically related to it. Also it prunes edges for pins related to too many boards (high degree). This results in more manageable graph size as well as more precise recommendations.

## 2.5 Graph Convolutional Network (GCN)

Graph Convolutional Network (GCN) is an algorithm that generates embeddings of graph structure of node neighborhood with the help of neural network. A variation of this algorithm is used by Pinterest for their recommendation system PinSAGE [17]. PinSAGE uses a hybrid version of this algorithm in which neural network is trained on the graph topology data as well as the features of the items. For better scalability the algorithm uses random walks to sample important neighbors of the target node. The trained neural network then produces embeddings of all items that can be used to make recommendations. The recommendations are made by searching for the nearest neighbors to the embedding of the query item. The main advantage of such algorithm is that it can use maximum of the information to make relevant recommendations.

## Conclusion

The problem of recommendations systems is not new and is well studied. There are number of traditional approaches used such as content-based recommenders and recommenders based on dimensionality reduction. The general problems with traditional methods are high computational and conceptual complexity and difficult scalability as well as "cold-start" problem when there is not enough information about the new item to give recommendations.

Lately, the new approaches to recommendations based on graphs have acquired interest of the researchers community. Some companies with large amount of content have already adopted graph-based recommendation systems. It has been shown that graph-based recommendations are viable alternative to the traditional methods. The literature offers graph-based recommendation algorithms that are in lesser degree dependent on the data size allowing them to scale well. Graph-based methods can easily be applied in systems with millions of items providing real time high quality personalized recommendations which is not feasible with the traditional methods. Such systems are conceptually simple and robust. With these advantages the problem of graph-based recommenders is interesting for research and the results of such research can potentially be applied in many companies.

## References

[1] "Netflix recommendations: Beyond the 5 stars." `https://medium.com/netflix-techblog/netflix-recommendations-beyond-the-5-stars-part-1-55838468f429`. Accessed on 19.06.2018.

[2] J. Bobadilla, F. Ortega, A. Hernando, and A. Gutiérrez, "Recommender systems survey," *Knowl.-Based Syst.*, vol. 46, pp. 109–132, 2013.

[3] J. Leskovec, A. Rajaraman, and J. D. Ullman, *Mining of Massive Datasets, 2nd Ed.* Cambridge University Press, 2014.

[4] W. Yang, Z. Wang, and M. You, "An improved collaborative filtering method for recommendations' generation," in *Proceedings of the IEEE International Conference on Systems, Man & Cybernetics: The Hague, Netherlands, 10-13 October 2004*, pp. 4135–4139, 2004.

[5] W. Carrer-Neto, M. L. Hernández-Alcaraz, R. Valencia-García, and F. G. Sánchez, "Social knowledge-based recommender system. application to the movies domain," *Expert Syst. Appl.*, vol. 39, no. 12, pp. 10990–11000, 2012.

[6] N. Zheng and Q. Li, "A recommender system based on tag and time information for social tagging systems," *Expert Syst. Appl.*, vol. 38, no. 4, pp. 4575–4587, 2011.

[7] L. Peska and P. Vojtás, "Enhancing recommender system with linked open data," in *Flexible Query Answering Systems - 10th International Conference, FQAS 2013, Granada, Spain, September 18-20, 2013. Proceedings*, pp. 483–494, 2013.

[8] B. J. Mirza, B. J. Keller, and N. Ramakrishnan, "Studying recommendation algorithms by graph analysis," *J. Intell. Inf. Syst.*, vol. 20, no. 2, pp. 131–160, 2003.

[9] B. M. Sarwar, G. Karypis, J. A. Konstan, and J. Riedl, "Item-based collaborative filtering recommendation algorithms," in *Proceedings of the Tenth International World Wide Web Conference, WWW 10, Hong Kong, China, May 1-5, 2001*, pp. 285–295, 2001.

[10] C. C. Aggarwal, J. L. Wolf, K.-L. Wu, and P. S. Yu, eds., *Horting Hatches an Egg: A New Graph-Theoretic Approach to Collaborative Filtering*, ACM, 1999.

[11] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web," in *Proceedings of the 7th International World Wide Web Conference*, (Brisbane, Australia), pp. 161–172, 1998.

[12] C. Eksombatchai, P. Jindal, J. Z. Liu, Y. Liu, R. Sharma, C. Sugnet, M. Ulrich, and J. Leskovec, "Pixie: A system for recommending 3+ billion items to 200+ million users in real-time," in *Proceedings of the 2018 World Wide Web Conference on World Wide Web, WWW 2018, Lyon, France, April 23-27, 2018*, pp. 1775–1784, 2018.

[13] A. Sharma, J. Jiang, P. Bommannavar, B. Larson, and J. J. Lin, “Graphjet: Real-time content recommendations at twitter,” *PVLDB*, vol. 9, no. 13, pp. 1281–1292, 2016.

[14] R. Lempel and S. Moran, “SALSA: the stochastic approach for link-structure analysis,” *ACM Trans. Inf. Syst.*, vol. 19, no. 2, pp. 131–160, 2001.

[15] A. Goel, A. Sharma, D. Wang, and Z. Yin, “Discovering similar users on twitter,” in *Proceedings of the Workshop on Mining and Learning with Graphs (MLG-2013)*, 2013.

[16] P. Gupta, V. Satuluri, A. Grewal, S. Gurumurthy, V. Zhabiuk, Q. Li, and J. J. Lin, “Real-time twitter recommendation: Online motif detection in large dynamic graphs,” *PVLDB*, vol. 7, no. 13, pp. 1379–1380, 2014.

[17] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, “Graph Convolutional Neural Networks for Web-Scale Recommender Systems,” *ArXiv e-prints*, June 2018.