Movie Recommendation Based on Graph Traversal Algorithms

Luboš Demovič, Eduard Fritscher, Jakub Kříž, Ondrej Kuzmík, Ondrej Proksa, Diana Vandlíková,
Dušan Zeleník, Mária Bieliková
Institute of Informatics and Software Engineering, Faculty of Informatics and Information Technologies
Slovak University of Technology, Ilkoviova, 842 16, Bratislava, Slovakia
televido@googlegroups.com

***Abstract*—Media content recommendation is nowadays a common problem. Traditional algorithms based on collaborative filtering require an up-to-date dataset of users and their preferences, which is difficult to gather for huge database of items. Content-based approach suffers from the complex computation of similarity among items. In this paper we propose an approach to recommendation with a focus on the natural change of user's interests in movies. We make use of a graph representation and experimented with modified graph algorithms. We design a representation of the data about movies in a graph structure and a method which uses our data model for recommendation. We propose four recommendation algorithms which are capable to find recommendations based on initial nodes, which selection is based on the user's current interests. We implemented these algorithms and experimentally evaluated them with real users.**

***Keywords*-recommendation; graph representation; graph algorithms;**

I. Introduction

In today's world there is a large amount of media content produced every day: movies, TV shows, TV programs etc. With so many options, choosing the right content can be difficult and overwhelming for the user. As for television, there are tens of channels from which the user can choose at any given moment. If he chooses the wrong channel, he misses something he would enjoy on a different one. The problem becomes even larger with more and more popular televisions or videos on-demand. The user has literally millions of possibilities from which to choose. The task becomes very difficult without a support which filters these possibilities.

Recommender systems analyze the taste, the mood or the context in which the user is at the moment. Based on the analysis, they create an accurate recommendation that suits the particular user [1]. There are various techniques used to create recommendations. The two main categories of recommendation systems are content based and collaborative. In both categories we should go through the entire entity base to find the correct item to be recommended. In collaborative approach this represents matrix of users and items, in content-based it is matrix of items and their similarity. Nevertheless, there is no guarantee that the estimation is correct and the recommended item is accurate enough for the user. Many recommendation systems try to recommend item by pairing the extracted knowledge base with the user's context and taste. As a result, the recommender systems may suffer from performance issues what makes them unusable in real time.

In this article we aim to design and evaluate a recommendation method that uses a new approach for recommending items with various attributes such as movies. Instead of the approaches mentioned above we decided to design a method which makes use of graph structure. We experimented with graph algorithms to provide comparison and their pros and cons. Graph-based recommender systems have been tested in the past and have shown promising results [2]. Our contribution is in comparison of graph algorithms and their modifications. We applied our method for multimedia recommendation (e.g. movies, TV shows and TV programs) in web application called Televido [1], which enabled us to experiment with various algorithms and real users.

II. Related Work

Recommender systems became very popular due to information overload which we have to face every day. There are approaches such as collaborative filtering or content-based [1]. Both approaches could be boosted by using graph representation to speed up the process of finding the most appropriate items.

In case of collaborative filtering we naturally need homogeneous bipartite graph representation since we are using single type connections among users and items. Mele et al. [3] proposed interesting solution using graph representation of direct connections among users. These connections express a behavior of these users. Connection among user u and v is added when user u discovered some item earlier than user v. This temporal modification of collaborative filtering is similar to well-known model of PageRank [4].

In case of content-based approaches we commonly use text optimized databases to quickly compute the similarity among items [5]. However, a graph could be one way to hold information on items. Especially if we use known metadata (categories, keywords, attributes) as the content to calculate similarity. A graph could be used as alternative to

[1]Televido - http://televido.tv/web

store similarity relations among items. This actually means that we need more connection types thus working with heterogeneous graphs. This is typical for multidimensional recommender systems as presented by Lee et al. [6].

Another option is to combine both collaborative filtering and content-based approach as proposed by Huang et al. [2]. In this work they proposed recommender system for digital library using Hopfield net algorithm. This work suggests to use graph algorithms to solve the problem of recommending items.

In the domain of media content recommendation research usually faces the problem calculating the similarity among items (in content-based). Authors often propose modifications of collaborative filtering. Biancalana et al. [7] used neural networks to train the model once. This actually reduced the multidimensional nature of their data but disabled them to make on-the-fly and up-to-date recommendations without retraining the model.

Another interesting work on using graph algorithm is presented by Chen et al.[8]. They used tripartite graph of users, items and queries used in the video search. This algorithm is very similar to spreading activation which could be promising in case of graph-based recommending. Similar approach was also presented by Bogers et al. [9]. In their work they used more contextual features (genre, language, director, actor, etc.). They used probabilistic algorithm which uses the matrix holding the intensity of transitions among entities while user browses them.

III. Representation of Movies and Metadata

To generate accurate recommendations we designed a graph structure which models the data - information about media content. Our proposed structure is based on RDF[2] triplet structure and inspired by the real life - how we talk or write about movies.

The main entities are movies, TV shows and TV programs, further referred to as just "movies". These are the items to be recommended. Other entities are people (actors, directors) and genres. These are represented by the nodes of the graph. The edges between nodes represent the attributes of the movies and they act as relationships between the entities. The resulting graph is actually one component (connected graph), which means that every movie must be indirectly connected to every other movie. Using real data the attributes ensure the connectivity and complexity of the model. For instance, every movie is associated to multiple genres. Since we have around 40 genres and around 165 thousands nodes in total we presume that the graph is one component. An example of the described data model graph structure is visualized in the Figure 1.

When creating the model we assumed that people also choose movies to watch by the actors or by their directors.

[2]Resource Description Framework - http://www.w3.org/TR/rdf-primer/

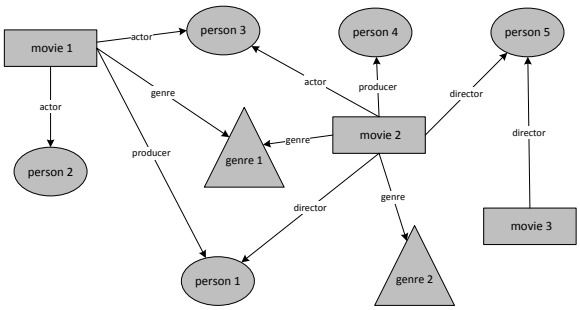

Figure 1. Example of the data model represented in graph structure.

In our approach this pattern is easily recognized and thanks to the model structure it is simple to determine the entities which should be recommended. This data model structure also ensures that we can find entities which are similar to other entities very quickly. This similarity stored implicitly in the connections among nodes.

IV. Recommendation Algorithms

Our recommendation algorithms are graph algorithms, which traverse the graph structure from initial nodes. The initial nodes represent user's interest. An initial node might be a movie or a genre the user likes. These initial nodes are either selected explicitly by the user as a query or implicitly based on the user model and the feedback from the user using standard methods. We actually do not need user model to be connected to nodes. We only need the list of items which are relevant for user and use it as a query on the fly.

The algorithms try to find the nodes which are the closest in the graph to all initial nodes, which are then returned as recommendations. There are multiple ways of looking at the problem of finding the closest nodes, especially in a very complex graph, which is why we designed and implemented four separate algorithms to compare.

Thanks to these parameters, the recommendation algorithms are quite versatile. For example, the same algorithm with different parameters can be used to recommend movies which are currently being shown at the cinemas or only the TV programs which will be on tomorrow night.

A. Union Colors Algorithm

The Union Colors algorithm is based on the basic *Breadth-first search (BFS)* graph algorithm. The algorithm works in the following way:

1) Mark each initial node with a different color (naturally, colors are represented by numbers).
2) Perform a simultaneous BFS from each initial node:
   a) Enqueue all initial nodes.
   b) Dequeue a node and visit it.
   c) Add all neighbors of the visited which are yet to be visited node into the queue.
   d) Repeat from step b)

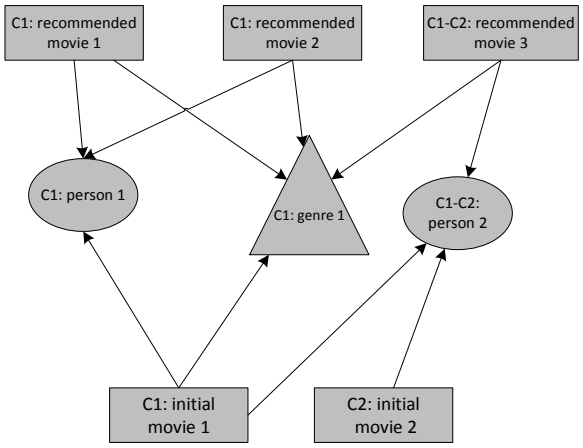

Figure 2. Visualization of Union Colors algorithm.

3) When a node is visited:
   a) If it is yet to be colored, color it with the color of the initial node
   b) If it is already colored (with a different color), merge the two colors into one - remember that one color equals the other
4) Keep merging colors until the last two colors are merged and only one color remains
5) Return the current and the next required number of nodes in the queue as the result

To merge the colors we can make use of the extremely fast *Disjoint-set data structure*, using which two colors can be merged in nearly constant time [10]. We actually do not need to traverse the whole graph. We only need to merge initial colors which count reflects the number of initial nodes.

Visualization of this algorithm is shown in Figure 2. Nodes which have the final color are going to be recommended. These are the nodes which come out as output of this algorithm. It actually means that nodes which are colored with final color earlier are going to be recommended as more relevant. *C1* and *C2* are the colors of initial nodes and the algorithm is stopped when it reaches the node "recommended movie 3".

B. Mixing Colors Algorithm

The Mixing Colors algorithm is similar to the previously described Union Colors algorithm. These two algorithms differ in the way they deal with the representation and the meeting of colors.

The algorithm works as follows:

1) Mark each initial node as visited by a different color.
2) Perform a simultaneous BFS from each initial node:
   a) Enqueue all initial nodes.
   b) Dequeue a node and visit it.
   c) When visiting a node, mark it with the colors of all the nodes which enqueued it.
   d) Try to add all neighbors of the visited node into the queue.
      i) If the node is not in the queue, add it and remember, that it was added by the visited node.
      ii) If the node is already in the queue, do not add it but remember that it has also been added by the visited node.
   e) Repeat from step b)
3) Continue until the required number of nodes is colored by every color, return them.

In Figure 3 the visualization of the algorithm is illustrated. The numbers in parentheses represent the colors of each node after one iteration of the algorithm.

C. Energy Spreading Algorithm

The Energy Spreading algorithm is based on *Spreading activation*, which is a method for searching associative networks, neural networks, or semantic networks. Similar to previous algorithms, it is a variation on the simultaneous BFS algorithm. The logic of the algorithm is following:

1) Set the energy of each initial node to some constant value.
2) Perform a simultaneous BFS from each initial node (as described in the previous section).
3) When a node is visited its energy increases by value E, $E = \frac{E_p}{n}$, where E_p is the energy of the parent node which enqueued the visited node and n is the number of nodes the parent node enqueued.
4) A node's energy can increase multiple times, but it only spreads it when it receives energy for the first time.
5) Continue until the required number of nodes is visited from each initial node.
6) Order the nodes by their energies and return the required number of the nodes, the more energy it has the higher it is.

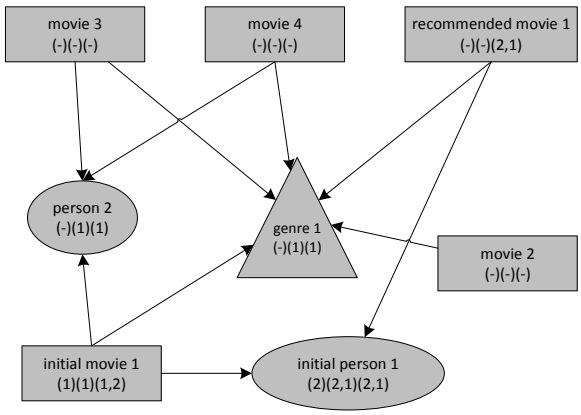

Figure 3. Visualization of the Mixing Colors algorithm.

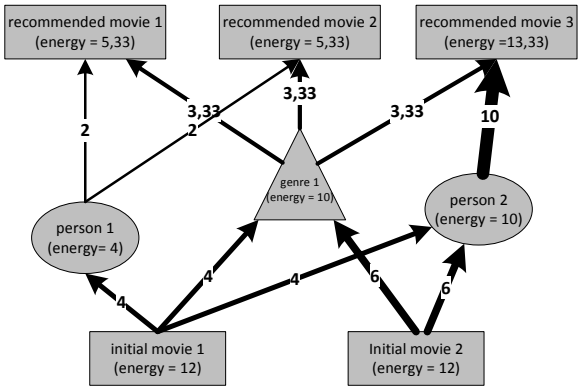

Figure 4. Visualization of Energy Spreading algorithm.

Visualisation of the algorithm is illustrated in Figure 4. The numbers on the arrows and their width represent the amount of energy which is being received by the receiving nodes. In the case shown in the figure, the movie number 3 is the first in the list of recommended nodes.

D. Modified Dijkstra's Algorithm

The Modified Dijkstra's Algorithm is, as the name suggests, based on the well known Dijkstra's algorithms for finding the shortest path in a graph. We implement a variation on this algorithm, which works for multiple initial nodes.

1) Run the Dijkstra's shortest path algorithm from each node to (some constant) maximal depth - calculate the shortest path from the starting node to each visited node.
   a) If the algorithm is ran from the first initial node, put all visited nodes with the value of the shortest path as their total value into a results set.
   b) If the algorithm is ran from other initial nodes, check if the node is in the results set. If yes, add the shortest path value to its total value.
2) Check if each node in the results set was visited from all initial nodes.
3) Order the nodes in the results set by their total values of the shortest paths.
4) Return the required number of nodes from the ordered results set.

The advantage of this algorithm is that it can be used on a graph with weighted edges. For our recommendation system it means that, for example, we can set all the edges which connect genres to movies to be longer than the edges which connect people to movies.

The logic of the algorithm is illustrated in Figure 5. The first two numbers in parentheses refer to the distance of each node to the initial nodes. The last value is their sum or *x*, which represents that the node was not visited from both initial nodes.

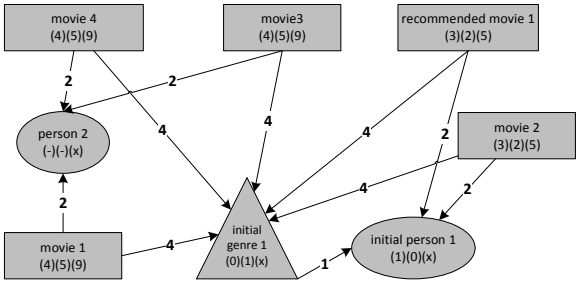

Figure 5. Visualization of Modified Dijkstra's algorithm.

V. Experimental Evaluation

The goal of the experimental evaluation we performed was to determine the best algorithm of the four algorithms we designed and to determine the accuracy of each algorithm. Based on the results of the experiment we intend to work on the most successful algorithms in the future and further modify them to make their results even better.

We filled the databases with a dataset consisting of information about a large amount of real movies and TV programs. In the end, the graph database consisted of roughly 165 000 nodes and 870 000 relations.

The experiment we used to test our methods was based on collecting explicit feedback from users. The user's task was to pick some initial nodes – movies, people or genres. The system generated 4 different sets of recommendations based on the initial nodes using the four recommendation algorithms we designed. We presented each user with 7 different scenarios, in each scenario the user was supposed to pick initial nodes in a different way, for example in the first scenario we asked him to pick two animated movies. The scenarios were implemented into the experiment to ensure the recommendation algorithms were tested on their versatility. After selection of the initial nodes each recommendation algorithm recommended five movies to the user; the user proceeded to order the algorithms based on their accuracy and was also asked to rate the most accurate algorithm on a scale from 1 to 5. In total, 30 users participated in the experiment. However, not all of the users who began the experiment completed every scenario. In total, 168 scenarios were completed by the users and recorded to our database, which, on average, works out to 5.6 completed scenarios per user.

The results of the experiment are shown in Table I. Here we can see comparison of the average ratings and positions. To discuss the results we need to add that Dijkstra and Energy Spreading are very expensive algorithms. These algorithms need relatively more time than other two proposed algorithms to compute results. This time could be above critical level in some cases. Especially energy spreading could last for very long time what would cause inappropriate waiting on the user side (in case we need real-time

Table I
EXPERIMENT RESULTS.

Algorithm	Average rating	Average position
Union Colors	2.322 out of 59	2.3095
Mixing Colors	2.9298 out of 57	2.2202
Energy Spreading	3.1892 out of 37	2.3095
Modified Dijkstra	3.1333 out of 15	3.1607

recommendation). The numbers in "Average rating" column mean the average rating from the cases when the algorithm was rated by the user (we only asked the users to rate the algorithm they picked as the first) and the number of times the algorithm was rated. "Average position" means the average position of the algorithms as sorted by the user, which means the lower the number the better.

The results show that the users usually picked the Union Colors algorithm as the first. This results might be biased - because of the way the experiment user interface was designed the first algorithm was always the same. Since the rating of the algorithm was the lowest, we assume the users left the first algorithm picked when they weren't very happy with the recommendation or were simply lazy.

Next in line are the Mixing Colors and the Energy Spreading algorithms. The former had a slightly better position but the latter had a better rating. These two algorithms appear to be the best of all, although they might require some improvement in the future.

The Modified Dijkstra's rating is not bad, but it was not picked as good nearly as often as other algorithms. The experiment we performed also had an auxiliary goal which was to find out information about the performance times of the algorithms. The only algorithm which had performance issues was the Modified Dijkstra's algorithm, which needs to keep a priority queue in order to work. The other algorithms' performance was satisfactory considering the scale of the data model graph, the recommendation was usually done in less than a second.

VI. CONCLUSIONS

In this paper we propose a graph-based recommendation method for recommending media content. We designed a graph-based data model and proposed four recommendation algorithms. The evaluation shows promising results, but it also shows the need to further research. Observing four different algorithms helped to achieve an improvement especially in the precision of recommendation.

The experiment showed that we can disregard the Modified Dijkstra's algorithm and probably also the Union Colors algorithm and focus on improving the rest. The data model can also be improved by instructing new types of relationships, for example keywords.

The main advantage of the graph representation for the task of item recommendation is performance. The implemented methods seem to be fast enough to work in real time. Our proposal is relevant for recommendation of huge number items and variable interests of users. Our approach is applicable also in other domains. Proposed traversal algorithms could be used in domains where we need to represent entities with variety of attributes and relations.

REFERENCES

[1] F. Ricci, L. Rokach, and B. Shapira, "Introduction to recommender systems handbook," in *Recommender Systems Handbook*. Springer US, 2011, pp. 1–35.

[2] Z. Huang, W. Chung, T.-H. Ong, and H. Chen, "A graph-based recommender system for digital library," in *Proc. of the 2nd ACM/IEEE-CS joint conf. on Digital libraries*, ser. JCDL '02. NY, USA: ACM, 2002, pp. 65–73.

[3] I. Mele, F. Bonchi, and A. Gionis, "The early-adopter graph and its application to web-page recommendation," in *Pro. of the 21st ACM int. conf. on Information and knowledge management - CIKM '12*. NY, USA: ACM, 2012, p. 1682.

[4] S. D. Kamvar, T. H. Haveliwala, C. D. Manning, and G. H. Golub, "Extrapolation methods for accelerating pagerank computations," in *Proc. of the 12th int. conf. on World Wide Web*, ser. WWW '03. NY, USA: ACM, 2003, pp. 261–270.

[5] M. Bielikova, M. Kompan, and D. Zelenik, "Effective hierarchical vector-based news representation for personalized recommendation." *Comput. Sci. Inf. Syst.*, vol. 9, no. 1, pp. 303–322, 2012.

[6] S. Lee, "A generic graph-based multidimensional recommendation framework and its implementations," in *Proc. of the 21st int. conf. on World Wide Web - WWW '12 Companion*. New York, New York, USA: ACM Press, 2012, p. 161.

[7] C. Biancalana, F. Gasparetti, A. Micarelli, A. Miola, and G. Sansonetti, "Context-aware movie recommendation based on signal processing and machine learning," in *Proc. of the 2nd Challenge on Context-Aware Movie Recommendation*. ACM, 2011, pp. 5–10.

[8] B. Chen, J. Wang, Q. Huang, and T. Mei, "Personalized video recommendation through tripartite graph propagation," in *Proc. of the 20th ACM int. conf. on Multimedia - MM '12*. New York, New York, USA: ACM Press, 2012, p. 1133.

[9] T. Bogers, "Movie recommendation using random walks over the contextual graph," in *Proc. of the 2nd Workshop on Context-Aware Recommender Systems*, 2010.

[10] M. Fredman and M. Saks, "The cell probe complexity of dynamic data structures," in *Proc. of the twenty-first annual ACM symp. on Theory of computing*, ser. STOC '89. New York, NY, USA: ACM, 1989, pp. 345–354.