A Proximity-Based Fallback Model for Hybrid Web Recommender Systems

Jaeseok Myung
« supervised by Sang-goo Lee »
Intelligent Data Systems Lab.
School of Computer Science and Engineering, Seoul National University, Seoul, Korea
{jsmyung, sglee}@europa.snu.ac.kr

ABSTRACT

Although there are numerous websites that provide recommendation services for various items such as movies, music, and books, most of studies on recommender systems only focus on one specific item type. As recommender sites expand to cover several types of items, though, it is important to build a hybrid web recommender system that can handle multiple types of items.

The switch hybrid recommender model provides a solution to this problem by choosing an appropriate recommender system according to given selection criteria, thereby facilitating cross-domain recommendations supported by individual recommender systems. This paper seeks to answer the question of how to deal with situations where no appropriate recommender system exists to deal with a required type of item. In such cases, the switch model cannot generate recommendation results, leading to the need for a fallback model that can satisfy most users most of the time.

Our fallback model exploits a graph-based proximity search, ranking every entity on the graph according to a given proximity measure. We study how to incorporate the fallback model into the switch model, and propose a general architecture and simple algorithms for implementing these ideas. Finally, we present the results of our research result and discuss remaining challenges and possibilities for future research.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Information filtering, Retrieval models*

General Terms

Algorithms, Design, Experimentation

Keywords

Recommender Systems, Fallback Model, Proximity Search

1. INTRODUCTION

In recent years, recommender systems (RSs) have received a lot of attention from both industry and academia. This interest has been spurred on by a number of commercial success stories in the field such as [15] and [5], and has led to a number of significant studies about effective recommendation models such as content-based filtering (CB), collaborative filtering (CF), and matrix factorization (MF) [1, 12].

One of main development issues facing the next generation of RSs is flexibility [1]. Most of the recommendation algorithms running RSs are hard-wired into the system by the system administrators and therefore support only a predefined and fixed set of criteria. However, the expansion of web services across several domains means that, more and more, different types of entities will co-exist within a single database.

The switch hybrid recommendation model [4] provides one possible solution to this problem. According to [4], a switching hybrid RS is one that selects a single recommender from among its constituents based on the recommendation situation. Although their study was limited to a single type of entities, we can easily apply this concept to our problem.

The integration of the switch model and the fallback model, however, provides a more complete solution. Algorithm 1 shows how a switch model could be used to select an appropriate RS according to its target type, such as movies, music, or books. However, if no RS exists that deals with a certain type of item, a fallback model is needed to complete the final *else* statement in the algorithm.

In this paper, we present several research topics and their possible solutions. Our main problems are as follows:

Algorithm 1 An example of a hybrid recommender system using the switch model and our fallback model. The switch model can be represented by if and $elseif$ statements. The notion of the fallback model is the final *else* statement.

Input: u, active user
Input: τ, target type of recommendation
RS = null
if τ = `movie` **then**
    RS = getMovieRS() ▷ domain knowledge
else if τ = `music` **then**
    RS = getMusicRS() ▷ domain knowledge
else if τ = `book` **then**
    RS = getBookRS() ▷ domain knowledge
else
    RS = **getFallbackRS()** ▷ fallback
end if
return RS.getRecommendationResults(u, τ)

WWW 2013 Companion, May 13–17, 2013, Rio de Janeiro, Brazil.
ACM 978-1-4503-2038-2/13/05.

- We study how to implement a fallback model for hybrid RSs. Our approach to this problem is to exploit a graph-based proximity search.
- We explore how to incorporate the fallback model into an existing system, with particular importance given to the need for the fallback model to only be considered if there are no existing RSs for a given item type. Our approach is to build a meta-hybrid model (named K-FIRST) between existing hybrid RSs and a fallback model that assigns higher priority to existing RSs.

To resolve the aforementioned challenges, we propose a graph-based recommendation framework named GraphRS (pronounced "graphers"). The rest of this paper will be spent describing the architecture and simple algorithms needed to implement these ideas and exploring experiment data that demonstrates the effectiveness of the proposed system.

2. RELATED WORK

In terms of a fallback model, few methods related to the one employed for this study exist because this paper is the first to employ the notion of a fallback model. A simple and intuitive means of implementing a fallback model is to always recommend popular items regardless of the target type. However, the popularity-based fallback model does not consider the user's personal preferences. In our experiments, this type of fallback model is employed as a baseline against which to judge the performance of our proximity-based fallback model in terms of recommendation accuracy.

Proximity search is a classic framework for ranking entities in a graph. A database can be viewed as a graph, with data in vertices (entities) and relationships indicated by edges. Therefore, we can compute the proximity of every pair of entities in the graph. Goldman et al. proposed a simple proximity measure that utilizes the shortest-path distance [7]. In our experiments, we implemented a representative existing proximity measure called Personalized PageRank (PPR) [16]. The PPR vector for a query vertex q can be calculated as follows:

$$\vec{r} = cT\vec{r} + (1-c)\vec{q} \quad (1)$$

where r_i is the score of node v_i, c is a damping factor constant that is normally given as 0.85, T is a transition matrix and $\vec{q}$ is a query vector where q^{th} element is 1 and the others are 0. PPR calculates the node authority value, but adjusts the score with a personalized bias. A bipartite graph-based example can be found in [14].

Actually, a number of methods exist for measuring proximities between entities in a graph. Jeh et al. [10] proposed a pairwise random walk-based ranking method, and Balmin et al. [3] presented a method to measure proximities between different types of entities in a database. In [18] and [13], paths on the graph given by administrators are used to their proximity measures. Recently, Shi et al. [17] conducted a study on the relevance in a heterogeneous information network.

These studies mainly focus on determining good proximity measures in order to build a new recommendation model. However, we concentrate on how to exploit those proximity measures as a fallback model. Therefore, our fallback model can incorporate all of the proximity measurement methods that we have mentioned.

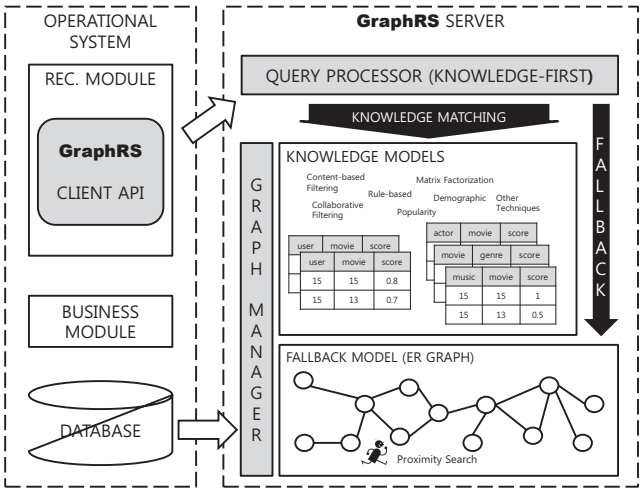

Figure 1: Architecture of GraphRS

3. PROPOSED APPROACH

In this section, we outline the architecture of the GraphRS system that incorporates the fallback model into an existing operational system.

Figure 1 provides a visual of the conceptual architecture of GraphRS. Businesses usually have an operational system which has a database and several business modules. One of business modules may be a recommendation module. GraphRS can be integrated with the existing recommendation module by a client API.

GraphRS consists of a client API and server system. The role of the client API is to send a query request and receive results. Most of the query processing steps take place in the server system. The client API is a wrapper for this process. A query is delivered to the server via HTTP requests and the server returns the results as a HTTP response. Therefore, the client and the server can both be placed in a single machine or distributed machines over a network, allowing the recommendation service to be provided as an independent system like a data warehouse.

The server system has two sub-modules, namely, GraphManager and QueryProcessor. GraphManager handles two responsibilities for the system: First, it analyzes the operational database in order to create an entity-relationship (ER) graph[1], which is used to copute the fallback proximity scores; and second, it manages knowledge models which refer to existing RSs. Therefore, knowledge models conceptually populate new types of edges on the ER graph.

QueryProcessor exploits both the knowledge models (existing RSs) and the fallback model (a proximity-based RS). Although the fallback model enables the recommendation of different types of entities, the accuracy of the proximity search is sometimes unpredictable. Knowledge models, on the other hand, can be optimized ahead of time, yet are only available for specific types of recommendations. Therefore, QueryProcessor assigns higher priority to knowledge models.

[1]In this paper, we distinguish between an ER graph and a heterogeneous graph. An ER graph refers to a graph created from the underlying database, while a heterogeneous graph refers to an extended ER graph having additional types of edges indicating different RSs.

We define the Knowledge-FIRST (K-FIRST) meta-hybrid model as follows:

$$K - FIRST(Q, v) = \sum_{q \in Q} (w(q) \cdot (K(q, v) + P(q, v))) \quad (2)$$

The K-FIRST model represents a recommendation score between user query Q and vertex v in the ER graph, where $Q = \{q|q \in V\}$ and $v \in V$. The weight of query entity q is determined by the function $w(q)$ where $\sum_{q \in Q} w(q) = 1$. $K(q, v)$ and $P(q, v)$ indicate scores generated by the knowledge model and the proximity-based fallback model respectively. More details on the K-FIRST model are discussed in Section 4.2.

4. METHODOLOGY

4.1 Graph Construction

We first explain how GraphManager converts the operational database into an ER graph. Figure 2 shows an example using the MovieLens dataset[2]. Let D be an operational database (a). We analyze foreign key constraints in D in order to create schema graph SG (b). Finally, we create an instance-level ER graph from SG (c).

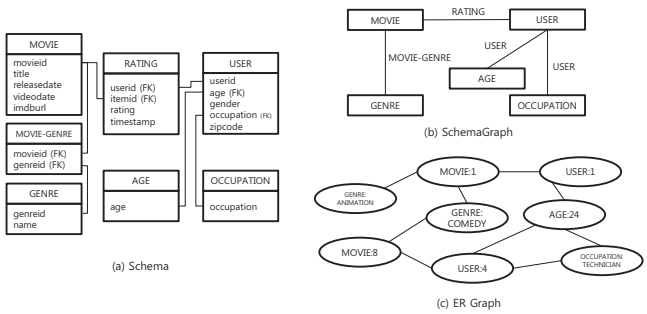

Figure 2: Example of Graph Construction Using the MovieLens Dataset

In (a), relations are connected by foreign key constraints. A foreign key consists of a referencing relation and a referenced relation. For example, RATING is a referencing relation, whereas USER and MOVIE are referenced relations. Referencing columns are marked (FK).

In (b), schema graph SG is derived from (a). We can obtain SG by registering a referenced relation as a vertex and a referencing relation as an edge. Algorithm 2 shows how to create a schema graph from a relational database. We first derive R_V and R_E from foreign key constraints in D. We also define function H from R_E to a power set of relations in D. This function returns a set of referenced relations to a given referencing relation. We then create an edge for each pair of relations in H(r).

It is worth mentioning that we include relation r in H(r) if the relation is an element of R_V (lines 5-7). This is because a relation in a database can be both a referencing relation and a referenced relation. In (a), we have USER, AGE and RATING relations. The relation AGE is only a referenced relation where as the relation RATING is only a referencing relation. However, the relation USER is both a referencing and a referenced relation. In this case, H(USER) contains only a relation AGE so the algorithm cannot make an edge between USER and AGE. This is why we need lines 5-7 in Algorithm 2.

[2]http://www.grouplens.org/

Algorithm 2 Creating a schema graph from a relational database

Input: R_V, a set of referenced relations in a database
Input: R_E, a set of referencing relations in a database
1: V=R_V
2: E={}
3: **for** each referencing relation r in R_E **do**
4: H(r) = a set of referenced relations of r
5: **if** r is an element of R_V **then**
6: Add r to H(r)
7: **end if**
8: **for** each pair of relations (r_1, r_2) in H(r) **do**
9: Add e=(r_1, r_2) to E
10: **end for**
11: **end for**
12: **return** SG=(V,E)

In (c), we finally obtain the ER graph. In this graph, each tuple in the referenced and referencing relations is translated into either a vertex or an edge respectively. For example, the graph shows us that "`user`:*1*" is a 24-year-old technician who watched "`movie`:*1*". We can obtain this graph by performing full table scans for all relations in D.

We applied this algorithm to the MovieLens-100K data set. The dataset contains 100,000 ratings of 1,682 movies from 943 users, with each user having rated at least 20 movies. In a schema graph, R_V contains the set of relations $\{movie, genre, user, age, occupation\}$. R_E includes the set of relations $\{movie - genre, rating, user\}$. As a result, the number of vertices in the graph is 2,732 and the number of edges is 210,416. The graph has symmetric edges. Experimental results on the graph are demonstrated in Section 5.

4.2 Query Processing

The GraphRS system allows system administrators to access several knowledge models. Hence, $K(q, v)$ can be computed using a weighted sum of an individual model score. In other words, $K(q, v) = \sum_{i=1}^{|M|} w(M_i) \cdot M_i(q, v)$ where $M_i(q, v)$ is a score on the edge $q \xrightarrow{type=M_i} v$. Weights can be determined by machine learning techniques such as [11], but this is out of scope of this paper.

Algorithm 3 K-FIRST hybrid model for knowledge and fallback models

Input: Q, a set of querie entities
1: K-FIRST(Q, v) = 0 for all $v \in V$
2: **for** $q \in Q$ **do**
3: **for** $v \in V$ **do**
4: $K(q, v) = \sum_{i=1}^{|M|} w(M_i) \cdot M_i(q, v)$
5: **if** $K(q, v) \neq 0$ **then**
6: $P(q, v) = PPR(q, v)$
7: **end if**
8: **end for**
9: K-FIRST(Q, v) += $w(q) \cdot (K(q, v) + P(q, v))$
10: **end for**
11: **return** $v \in V$ in descending order of K-FIRST(Q, v)

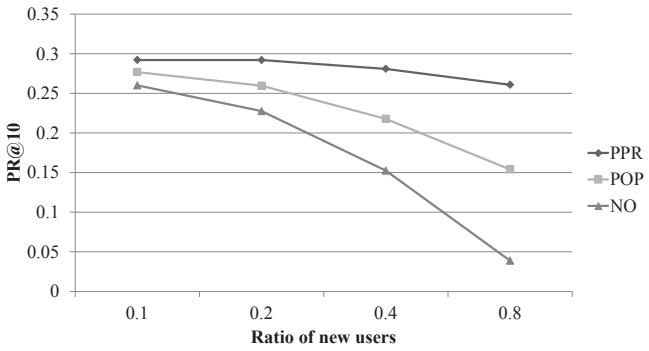

Figure 3: Comparison of Fallback Models Applied to the New User Problem

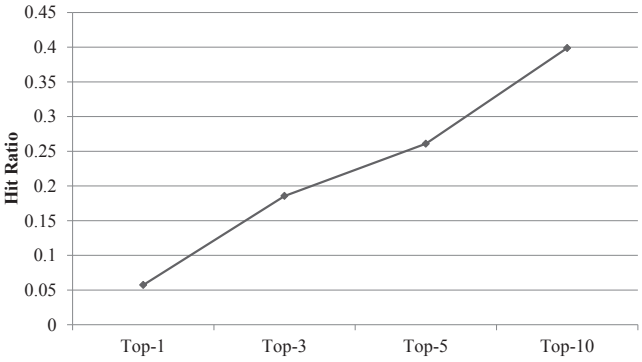

Figure 4: Example Application to Identify Users Using the Proximity-based Fallback Model

K-FIRST assigns higher authority to knowledge models than it does to the proximity model so that, for a single q in Q, the proximity cannot change rankings determined by the knowledge model. However, the proximity model can contribute to an overall ranking that takes all query entities and their weights into consideration. This derives the following definition of $P(q, v)$, that $P(q, v) = 0$ if $K(q, v) \neq 0$, otherwise $P(q, v) = PPR(q, v)$.

Given a set of query entities Q, Algorithm 3 shows how a system obtains the hybrid recommendation score for all $v \in V$. For each q, we evaluate $K(q, v) + P(q, v)$, and finally obtain the weighted sum of the hybrid results.

Online Top-k Query Processing: Efficient top-k computation is also important for the K-FIRST model because RSs have to deal with hundreds of thousands of recommendation requests. Each request consists of one or more query entities, and each of them can be involved in several knowledge models. Let us suppose that each knowledge model has its own top-k mechanism, which implies that we have several sorted lists that contain the results of knowledge models. The computation of top-k $K(q, v)$ can then be viewed as a traditional top-k query processing problem as is familiar in the database research field [9]. The NRA (No Random Access) algorithm [6] in particular may provide a solution to our problem. Top-k PPR algorithms have also become available as of late for $P(q, v)$ [8, 2]. Finally, the computation of $K(q, v) + P(q, v)$ can be solved by nested NRA.

5. RESULTS

We will now evaluate the effectiveness of the proposed approach with several experiments.

5.1 Comparison of Fallback Models

First of all, we show how the proximity-based fallback model helps an existing RS. The new user problem is a typical scenario where a hybrid model can be useful. We prepared a CF-based knowledge model using a partial set of users and assumed that this is the only knowledge model that we have. We then use the model if it is available, but otherwise several fallback models are employed. Therefore, the importance of the fallback models increases as the ratio of new users increases.

In Figure 3, we examine three fallback models: 1) no fallback; 2) popularity-based fallback (POP); 3) proximity-based fallback (PPR). The graph shows that the precision of the no-fallback model decreases sharply while those of POP and PPR models fall off much more slowly. Moreover, PPR outperforms the POP scheme because POP only produces a static list of movies. This experiment assumes only a user-movie recommendation service. However, in reality there are tens of thousands of different web services in use. Therefore, the fallback model is most useful when a RS needs to support several recommendation services.

5.2 Applications Using Fallback Models

The proximity-based fallback model has numerous applications. For instance, we can identify an active user by using a set of movies that the user already has seen. This implies that an online shopping site can characterize a guest user by examining a short sequence of item page views. In this experiment, we assume that there are no prepared knowledge models for this problem. Therefore, the performance totally depends on the proximity-based fallback model.

To conduct the experiment, we first delete the active user's entire ratings history and then compute the PPR of all movies and registered users. After that, we issue a query consisting of five movies which were in the active user's ratings. We determine whether an active user is hit or not by examining whether the user appears in a top-k list. This process is repeated for all 943 users in the dataset. The experimental result represented in Figure 4 shows that almost 40% of users are hit in top-10 out of 943 users. Even though the fallback model is not the best solution for a specific problem, it handles most problems reasonably well.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we proposed a proximity-based fallback model for existing hybrid RSs. The necessity of fallback models was shown by giving motivating examples and referencing existing research. We presented the GraphRS system that utilizes the fallback model. In the system, existing RSs are adopted as knowledge models, and the knowledge models are integrated with the fallback model using a meta-hybrid (K-FIRST) model.

The fallback model choice is an important aspect of our approach. One of the simplest solutions is to use a popularity-based fallback model. However, popular items are hard to personalize, so we instead exploit graph-based proximity search techniques as a fallback model.

We expect further research will lead to significant improvements. Still, there are remaining issues that we plan to tackle in future work as follows:

Learning to Rank for K-FIRST: Different weights on query entities and knowledge models should be considered in order to improve recommendation accuracy. For instance, collaborative filtering and matrix factorization techniques can be integrated according to their weights. Sometimes, the end-user may be able to select a specific knowledge model that satisfies their requirements. Different weights on query entities are also needed. For example, user entities should be assigned a high weight in order to produce personalized recommendation results.

Novel Proximity Measure and Efficient Top-K Processing on a Large Graph: The notion of a proximity search is very general. It sometimes refers to the similarity between entities, while at other times referring to the distance between entities. For the fallback model, a proximity measure that maximizes the average precision among several domains would be a good solution. In addition, the computation of a measure has to be efficient.

Recommender Systems into Data Warehouse: The architecture of GraphRS enables the system to be operated independently with other business modules. This is similar to the data warehouse that supports intelligent user services. We believe that integration between a RS and a data warehouse will create a synergistic effect.

7. ACKNOWLEDGEMENT

This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MEST) (No. 20110030812)

8. REFERENCES

[1] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans. on Knowl. and Data Eng.*, 17(6):734–749, June 2005.

[2] K. Avrachenkov, N. Litvak, D. Nemirovsky, E. Smirnova, and M. Sokol. Quick detection of top-k personalized pagerank lists. In A. Frieze, P. Horn, and P. Pral'lat, editors, *Algorithms and Models for the Web Graph*, volume 6732 of *Lecture Notes in Computer Science*, pages 50–61. Springer Berlin Heidelberg, 2011.

[3] A. Balmin, V. Hristidis, and Y. Papakonstantinou. Objectrank: authority-based keyword search in databases. In *Proceedings of the Thirtieth international conference on Very large data bases - Volume 30*, VLDB '04, pages 564–575. VLDB Endowment, 2004.

[4] R. Burke. The adaptive web. chapter Hybrid web recommender systems, pages 377–408. Springer-Verlag, Berlin, Heidelberg, 2007.

[5] A. S. Das, M. Datar, A. Garg, and S. Rajaram. Google news personalization: scalable online collaborative filtering. In *Proceedings of the 16th international conference on World Wide Web*, WWW '07, pages 271–280, New York, NY, USA, 2007. ACM.

[6] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '01, pages 102–113, New York, NY, USA, 2001. ACM.

[7] R. Goldman, N. Shivakumar, S. Venkatasubramanian, and H. Garcia-Molina. Proximity search in databases. In *Proceedings of the 24rd International Conference on Very Large Data Bases*, VLDB '98, pages 26–37, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[8] M. Gupta, A. Pathak, and S. Chakrabarti. Fast algorithms for topk personalized pagerank queries. In *Proceedings of the 17th international conference on World Wide Web*, WWW '08, pages 1225–1226, New York, NY, USA, 2008. ACM.

[9] I. F. Ilyas, G. Beskales, and M. A. Soliman. A survey of top-k query processing techniques in relational database systems. *ACM Comput. Surv.*, 40(4):11:1–11:58, Oct. 2008.

[10] G. Jeh and J. Widom. Simrank: a measure of structural-context similarity. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '02, pages 538–543, New York, NY, USA, 2002. ACM.

[11] T. Joachims. Training linear svms in linear time. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '06, pages 217–226, New York, NY, USA, 2006. ACM.

[12] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, Aug. 2009.

[13] S. Lee, S. Park, M. Kahng, and S.-g. Lee. Pathrank: a novel node ranking measure on a heterogeneous graph for recommender systems. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, CIKM '12, pages 1637–1641, New York, NY, USA, 2012. ACM.

[14] S. Lee, S.-i. Song, M. Kahng, D. Lee, and S.-g. Lee. Random walk based entity ranking on graph for multidimensional recommendation. In *Proceedings of the fifth ACM conference on Recommender systems*, RecSys '11, pages 93–100, New York, NY, USA, 2011. ACM.

[15] G. Linden, B. Smith, and J. York. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1):76–80, Jan. 2003.

[16] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab, November 1999. Previous number = SIDL-WP-1999-0120.

[17] C. Shi, X. Kong, P. S. Yu, S. Xie, and B. Wu. Relevance search in heterogeneous networks. In *Proceedings of the 15th International Conference on Extending Database Technology*, EDBT '12, pages 180–191, New York, NY, USA, 2012. ACM.

[18] Y. Sun, J. Han, X. Yan, P. S. Yu, and T. Wu. Pathsim: Meta path-based top-k similarity search in heterogeneous information networks. In *In VLDBąŕ 11*, 2011.