UNIVERSITY OF CINCINNATI

Date: 08/15/2008

I, Svetlana Strunjas,

hereby submit this work as part of the requirements for the degree of:

Ph.D.

in:

Computer Science

It is entitled:

Algorithms and Models for Collaborative Filtering from Large Information Corpora

This work and its defense approved by:

Chair: Dr. Fred Annexstein

Dr. Kenneth Berman

Dr. Karen Davis

Dr. Kevin Kirby

Dr. John Schlipf

Algorithms and Models for Collaborative Filtering from Large Information Corpora

A dissertation submitted to the

Division of Research and Advanced Studies
of the University of Cincinnati

in partial fulfillment of the
requirements for the degree of

DOCTOR OF PHILOSOPHY

in the Department of Computer Science
of the College of Engineering

August 2008

by

Svetlana Strunjaš
B.S., University of Belgrade, 2000, Belgrade, Serbia

Dissertation Adviser and Committee Chair:
Fred S. Annexstein, Ph.D.

Abstract

In this thesis we propose novel collaborative filtering approaches for large data sets. We also demonstrate how these collaborative approaches can be used for creating user recommendations for items, based upon preferences towards items that users demonstrated in the past.

We propose a framework, called a *collaborative partitioning* or CP for short, that is focused on finding a partition of a given set of items in order to maximize the number of partition-satisfied users. Two theoretical models for evaluating the quality of partitions are proposed. Both are introduced as bicriteria optimization problems with the percentage of satisfied users and the level of users satisfaction as the two optimization coefficients. As both of these bicriteria optimization problems are NP-hard, we propose Hierarchical Agglomerative Clustering - based approaches to compute approximations of their solutions. The results obtained by running the heuristic approaches on a real dataset show that the proposed approaches for CP have good results and find items partitions that are very close to a human-based genre partition for a given set. The genre partitions are partitions of items according to some human-created classifications. The results also show that the proposed heuristic approaches are a very good starting point in creating a top-k recommendation algorithms.

The second part of this thesis proposes a collaborative filtering framework for finding *seminal* and *seminally affected work* for sets of items.
The concept of *seminal work* for a set of items is used to mark items released in the past that are highly correlated to some future sets of items in the terms of users preferences. Similarly, the *seminally affected work* is a concept that is used in this thesis to mark items that are highly correlated to some previously released (older) items in the terms of users preferences.
In this approach, we translate item-item correlation into a correlation directed acyclic graph (DAG). Direction in the DAG is determined by a chronological ordering of items. We demonstrate and validate the proposed approach by applying it on the web-based system called MovieTrack. This system uses *seminal* and *seminally affected work* in movies to give movie recommendations to users. It is built by applying the previously proposed approach on a real data set of movie reviews released by Netflix.

Acknowledgements

This dissertation is dedicated to the loving memory of my father,
Spasoje Strunjaš

Throughout my graduate school journey at the University of Cincinnati, there were so many people who influenced my life and my scholarly experiences. I was very fortunate to be surrounded by wonderful and very bright people. I would like to thank to my academic adviser, Dr. Fred Annexstein and co-adviser Dr. Ken Berman. Dr.Annexstein spent many hours having brainstorming discussions with me and has greatly influenced my scientific and critical way of thinking, and I will always be very grateful to him. His guidance, encouragement and support were very valuable to me. Dr.Berman also spent many hours brainstorming with Dr.Annexstein and myself, discussing correctness of models and theorems and finding very valuable counter examples for some cases that were far from obvious. Besides providing me with their extensive knowledge and expertise in computer science theory, Dr.Annexstein and Dr.Berman have been very good friends to me and to other people in our lab. We have had wonderful time going together to conferences and, throughout my years at the University of Cincinnati, I have met their wonderful families and spent time with them as well.

I would also like to thank to Dr. John Sclipf, Dr. Karen Davis and Dr.

Kevin Kirby for being my committee members, and for dedicating their time to read my dissertation and to provide me with their academic guidance.

As a graduate student at the University of Cincinnati I have spent many days and hours researching in my lab. Thanks to my lab mates, these days were easier to bear. Many thanks to my lab mates Kevin and Aravind, for brainstorming with me, helping me solving some technical and coding problems that I have encountered during my research, having coffee with me and simply for being my friends. Besides Kevin and Aravind, I was fortunate to meet some other very nice people here in Cincinnati and they became very good friends of mine. I would like to thank to Irena and Emily for their sincere and beautiful friendship.

My special thanks go to my mother Paula, who, although far away in my home country, has always been there for me. There are so many things that Mom has done for me – shared my problems and joys throughout this journey, lived through every moment together with me, sacrificed so much so I can live better, provided me and my husband with so much love and care. She is one incredible person, and I hope I have inherited at least a part of her goodness and her loving personality.

I would like to give thanks to my late father Spasoje who is not here with us anymore, but will always live in my heart. My Dad has influenced my curiosity for science and mathematics since the very first beginnings, and I

know he would be so proud of me now. Although he is not here with us anymore, his legacy of goodness and honesty and his scientific pursuits will be continued through his children - my brother and me.

Although far away from my home country, I have been fortunate enough to have a part of my family living here, in the Cincinnati area. I would like to thank to my brother Saša, his wife Jean and their family for their love, support and care. Saša has always reminded me of my homeland, and thanks to him I have managed to keep my family memories and traditions alive and refreshed. Special thanks go to my adorable nieces Anna and Maria, for giving me so much joy and happiness in watching them grow up.
I would also like to thank to my extended family for their love, support, care and for everything they have done and provided for my husband and me. Sister-in-law Nicolle and her husband Fred, little nephew Jakob, father-in-law Marvin, mother-in-law Shirley, Aunt Frances and many other wonderful people have brought so much joy and happiness to me and my husband throughout these years.

Finally, I would like to thank to my husband Chad Yoshikawa, for his love, support and companionship as well as his help with coding problems and brainstorming sessions that we have had. He has always been there for me and he has enriched my life in so many ways. Chad has given me so much support through tough times, and he has shared with me joy and happiness

of good times. I am very fortunate and grateful for having such a good, loving and big-hearted husband and am looking forward to our future life together.

Table of Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Information Overload in the Petabyte Age Finding relevant information has been a challenge for a long time. Six decades ago people started reading digital information using computers. Two decades ago the Internet made digital information accessible to more people. More than a decade ago the first web search engines (WebCrawler and Lycos in 1994) started crawling Internet information and storing it in a single database. Today Google servers process a petabyte of data every hour [34].

Information filtering is the process that removes information that is not of interest to users. The majority of information we are browsing through and choosing from today is already filtered even before it becomes available for our viewing. If we did not have any kind of filtering when browsing for relevant information, even the simplest tasks of finding the right information would be a long and possibly not a very successful process. For example, when we

log into the Blockbuster website, it automatically filters some movie items that we might like, based on our viewing history. When we are searching for news at some of the popular news websites, such us Google News [12], we can create a personalized view of news items that match our interests. Based upon our current interests and viewing history, the system will filter all news items that might be interesting to us and help us to easily browse through the massive amount of news available every day. Another example of information filtering can be found on the Amazon website. When we log into the Amazon website, it shows us some catalog items that, most likely, might be of interest to us, based on our purchasing and rating history.

The recent years have increased information filtering challenges by introducing the new "Petabyte Age", where useful and relevant information are filtered for users from petabytes of information stored in the Internet cloud. The amounts of information we are exposed to nowadays are massive and far beyond our comprehension and ability to digest. Just for example, The Amazon.com company has had several million available catalog items and 29 million active customers in 2003 [15].

That number today is significantly higher. Netflix recently released their ratings dataset that contains more than 100^6 movie ratings for 17770 movies rated by 480189 users [23]. The size of the data set that Netflix processes everyday is believed to be much larger than that. StumbleUpon [29], one

of the largest social media sites, as of April 2008, contains over 5 million users that gave over 5 billion ratings (called "stumbles") for pages that they found on the Web [33]. Google News website keeps track of several million news articles from 4500 news sources throughout the world. Looking at these statistics we can see that filtering information at a user's level is not enough anymore. The massive size of information on these websites today requires that we establish concise and mathematical models to be able to search and retrieve efficiently. The Petabyte Age introduces great challenges of creating new models and automated approaches for making information filtering scalable and precise at the same time.

In this thesis we present novel models and scalable algorithms for filtering information from large information corpora. The main focus of this thesis is in the specific area of information filtering called collaborative filtering (or social information filtering). A detailed introduction to collaborative filtering is given in the second chapter. Our objectives in this thesis are in designing collaborative filtering approaches that can be used to imply future user interests based upon existing users preference data. The problems that we are attempting to solve when designing our collaborative filtering models and algorithms are two-fold and tightly connected - scalability (with respect to the time complexity) and accuracy (with respect to user satisfaction). When dealing with massive information corpora, scalability of the algorithms applied to the corpora is of a crucial importance. Small changes in the time complexity can result in significant computational overheads and cause a pro-

gram to run, instead of a couple of hours, several days, even weeks or months. The second important issue in CF area is accuracy. A collaborative filtering system has to be as accurate as possible, because we do not want to filter out and "hide" from a user items that might be interesting to him. From the other side, we also do not want to overwhelm the user with a large number of items that are not interesting at all, because that completely defies the purpose of information filtering. Sometimes optimal solutions in the terms of accuracy for problems in collaborative filtering are NP-hard, as it is for the bicriteria optimization problems related to the collaborative partitioning (CP) framework proposed in this thesis. This motivates heuristic approaches to the problems.

For a given set of users U and items I that are rated by the users, CP focuses on the problem of identifying item partitions that optimize the fraction of users that will be satisfied with a given partition. Unlike the traditional clustering, the proposed CP model is a novel, bicriteria optimization problem, which has a goal of finding the best clustering that satisfies the most users at the highest level of satisfaction. CP of items is used to identify clusters of users with same or very similar taste towards the given sets of items (they like or dislike items in a same way). Besides that, it is also used to identify possible existence of preferences towards certain genres of movies in a given dataset. Another use of CP is to generate, from a given user-items preference data, a hierarchy of topics that assists users to efficiently

browse through a massive set of items to find items of a possible interest. CP model can be further extended, as it is done in this thesis, to produce top-k recommendation algorithms.

One of the ways to find scalable algorithmic solutions for bicriteria CP problem, and problems with the same computational complexity as the CP problem is to find heuristics that approximate the optimal solution.

The first part of this thesis introduces our mathematical model and approach for collaboratively finding topic segmentation of items to maximize users' satisfaction. This problem is NP-hard (for proofs and discussions see 3.4) and our approach in solving it is to find suitable heuristics that approximate the optimal solution.

In the second part of this thesis we propose a novel collaborative filtering framework for finding *seminal* and *seminally affected work* for sets of items. The concept of *seminal work* for set of items is used to mark items released in the past that are highly correlated to some future sets of items in the terms of users preferences. Similarly, the *seminally affected work* is a concept that is used in this thesis to mark items that are highly correlated to some previously released (older) items in the terms of users preferences.

In this approach, we translate item-item correlation into a correlation directed acyclic graph (DAG). Direction in a DAG is determined by a chronological order of items, *e.g*, in DAGs that are used for finding *seminal work* for a given set of items, edges are directed from older to more recent items,

and in DAGS that are used for finding *seminally affected work* for a given set of items, edges are directed from more recent to older items. Querying for *seminal* and *seminally affected work* for given sets of items is achieved by querying previously described DAGs for ancestor queries. Querying DAGs for ancestor queries is implemented in efficient and scalable way by designing compact labelling schemes for nodes in DAGs that contain information about structural properties of DAGs. By applying our proposed labelling schemes we are able to encode all-path information for each node (item) in correlation DAGs by labelling each node with a relatively short labels and evaluating ancestor queries for a set of items by directly reading structural information encoded in their labels.

We demonstrate and validate the proposed approach by applying it on the web-based system called MovieTrack. This system uses *seminal* and *seminally affected work* in movies to give movie recommendations to users. It is built by applying the previously proposed approach on a real data set of movie reviews released by Netflix.

Both of the described approaches are focused on mathematically representing users preference data and using these models to infer future preferences for users based on their ratings history.

The contribution made in this thesis is two-fold - we proposed mathematical approaches for modeling users preference in large data sets and we also designed algorithms and methodologies that can be applied for testing the proposed approaches on real data sets.

Chapter 2

Background Work and Terminology

When discussing methods in information filtering, there are three dominant research paradigms and approaches – *content–based filtering*, *collaborative filtering* (also known as *social information filtering*) and *hybrid filtering*.
In this chapter we give an overview of existing information filtering methodologies, describe existing systems that that implement the mentioned methodologies and give comparisons with research proposed in this thesis.

2.1 Content-based Information Filtering

Content-based information filtering exploits only information that can be derived from the content of filtered objects. This kind of information filtering

is closely related to the area of information retrieval, and it has many similarities with it. Both approaches are usually applied on the textual data, although content-based information filtering usually deals with semi-structured and unstructured data, whereas information retrieval most commonly deals with structured data. Many information retrieval research methods and results are used to accomplish content-based information filtering tasks. This holds especially for IR methods in the areas of text representation (indexing), retrieval techniques and evaluation of retrieved results. However, besides these similarities, content-based information filtering has some distinct characteristics that separate it from information retrieval as a well defined and unique area. Information retrieval is usually concerned with the selection of texts from a relatively static database, while content-based information filtering is mainly concerned with selection and elimination of texts from dynamic data streams. Content-based information filtering is usually based on description of individual or group preferences, often called profiles, which are gathered and accumulated during longer time periods, while information retrieval is focused on retrieving information relevant to a user's one-time need for knowledge. That need is usually described with a query that is written in a system-specific query language, and it is executed only once or couple of times. Content-based information filtering is concerned with long-term information goals of its users and repeated use of a system, while information filtering is usually concerned with single uses of the system by a person with a one-time goal and one-time query. Complete description of similarities and

differences of these two areas can be found in the Belkin *et. al*, 1992 [5].

Content-based information filtering is based on selecting items for a given user based upon correlations between the content of unseen items and the user's preferences in the past. For a given user and the user's profile, these systems are trying to find future items of interest for the user by measuring similarities between the contents of the user's preferred items in the past and unseen items and filtering out all unseen items which content is not highly similar to the content of items in the user's profile. The following systems are applying content-based information filtering when presenting items to a user.

2.1.1 Google News

Google News website is a news site that aggregates news articles from more than 4500 news resources worldwide, groups similar stories together and displays them according to each reader's personalized interest. Besides filtering news items according to personalized interests, this system also provides news alerts, where a reader chooses keywords of interest and the system automatically filters incoming news that match the given keywords and sends them to the reader's email. As it was mentioned before, this system keeps tracks of several million news articles, and without the information filtering functionalities provided by the Google News, filtering out non-relevant news items would be a very tedious process for users.

2.1.2 SIFT

Stanford Information Filtering Tool (SIFT) [35] is a content-based information filtering system that started as a small experimental service at the Stanford University in 1994, grew to a reasonably large system with 18400 daily users and 40100 long-term profiles within few years, and eventually became a commercial operated system operated by the Sift Inc. startup company. SIFT collects Usenet NetNews and articles from different mailing lists, and gathers users' profiles using email of web forms. Every user submits a description of its own profile using some forms of information retrieval languages. At the beginning stages of the SIFT development simple Boolean queries were used to describe profiles. Afterwards, when complexity of descriptions for user profiling increased, the SIFT started to utilize more complex query languages and implemented a new indexing scheme for profiles, capable to handle Vector Space Model (VSM) [26] queries. When a user logs into SIFT, the system tries to do optimal matching between all incoming news articles and the user's profile description. At the beginning stages of SIFT development, all item-profile matchings were performed on a centralized server, but after some time scalability became a huge issue, so matching process was distributed on multiple servers.

2.1.3 NewsSieve

NewsSieve [13] is a content-based information filtering system that was used for content-based filtering of the Usenet news groups. It was developed as a part of a research project at the University of Bonn, Germany. Each user in the NewsSieve system starts building its profile by reviewing unfiltered messages and marks interesting and unimportant text. All documents in the NewsSieve are reduced to their words and their frequencies. In the beginning stages of the NewsSieve development all rated documents from profiles as well as all documents from the NewsSieve system were represented as vectors using Vector Space Model (VSM). For a given user, the output of the filter was calculated by searching for new messages that were represented by a vector that matched the user's profile vector. The advantage of this approach was the fact that filters based on the VSM model were easy to implement. However, this approach that performed VSM- based filtering did not achieve a very high accuracy, so the vectors were refined into a rule system. Each profile was redesigned to contain a set of rules that assigned a certain scores not to just one word but to groups of words. The total score of a text is computed as a sum of all assigned scores, for a given profile.

As previously mentioned, content-based information filtering is applicable only to filtering objects that have parsable content, such as text documents and web pages, or images, videos or other multimedia items that are manually tagged with some text (*e.g.*, "image with red flowers", "video of the Eiffel Tower" *etc.*). More recently a couple of projects evolved that are dealing

with image filtering based upon analyzing and comparing contents of images - color, texture and shape, but they are dealing with digital image processing algorithms and are outside of scope in this thesis.

The main drawback of this approach, besides the fact that it is applicable only to items with parsable content, is the fact that it usually cannot deduce any kind of hidden, non-transparent preference correlation information between different users and different items.

2.2 Collaborative Filtering

Collaborative filtering (also called Social Information Filtering) analyzes similarities in taste between different users to filter information items. It relies on the fact that users' preferences are not randomly distributed – when analyzing item ratings given by large groups of users, we can usually detect general trends and patterns within the taste of one person as well as between groups of people.

In collaborative filtering systems, each user rates certain subset of information items. The rating scale varies from one dataset to another. For example, in the StumbleUpon social media site, each user can rate web pages and multimedia files that they found on the Web by either given them a "thumb up", which means that the user likes the item, "thumb down", which means that the user dislikes the item, and "no opinion". In the Cinematch system [6] developed by Netflix, users rate movie items on the scale from one to

five stars, and the most positive opinion is expressed with five stars. Based upon existing preferences of a given user, a social information filtering system filters information about the interests of the user by collecting taste information from many other users (collaborating).

A simplified example for social information filtering can be given on the next movies example. Let's assume that two users A and B watched many similar movies and have very similar taste patterns (*e.g.*, they are both huge science fiction and war movies fans). Assume that both users watched and liked some science fiction movies, such us "Blade Runner", "2001 A Space Odyssey", "Star Wars I", "Star Wars II", "Star Wars III" and "Star Wars IV", as well as some war movies, such as "Saving Private Ryan", "Enemy at the Gates" and "Schindler's List". From this data we can, with a certain probability predict that the user A will most probably like "Star Wars IV" or "Patton" if the user B has already seen and liked those movies. This simplified approach can be also applied for movies that one of these users watched and didn't like and other user hasn't seen yet. The previous example is just one form of social information filtering. Besides grouping users in similar groups according to similarities between their common preference patterns, we can also group movies (items) in a given dataset according to some topics-based classification that is defined by the given dataset. In that way we can detect subsets of items that are usually preferred together by majority of users. The following systems are examples of academic collaborative filtering projects.

2.2.1 GroupLens

GroupLens [18], [14] is a collaborative filtering system designed for filtering Usenet news at the University of Minnesota. The goal of this system is to help people to collaborate on finding articles they will most probably like in incoming streams of available Usenet articles. The architecture of this system is designed to blend into existing Usenet client/server architecture. User logs into the system using the client newsreader application. This application then connects to the user's local Usenet news server and to the GroupLens server. For each user the GroupLens server contains information about all ratings the user has made in the past. Besides that, this server contains, for each article all ratings that are given by users for that article. Based on the rating values for the given user, the system identifies users that are most similar to the current user, filters out news articles retrieved from the Usenet news server, and presents to the given users only articles that are highly ranked by the most similar users. The current user then browses through the retrieved articles, rates them on the scale from one to five and sends back his rating updates to the GroupLens server.

2.2.2 MovieLens

MovieLens [10], [19] is an online movies recommender system developed at the University at Minnesota, in the same research group that developed the previously described GroupLens system. It invites users to rate movies

that are contained in their database, and in return makes personalized recommendations and predictions on movies that the user has not yet rated. This online service is one of the most popular non-commercial movies recommendation websites. Each user that wants to rate movies on the MovieLens website is asked to create a user account, where all personal preferences and movie ratings for the given user are recorded. Movies are rated on a rating scale from 1 to 5, with 5 being the highest score. Seven years ago, the MovieLens research group released their ratings dataset, that contained 1000029 movie reviews given by 6040 MovieLens users for approximately 3900 movies. Today the size of their data set is much larger.

2.2.3 Ringo

Ringo [27] is a collaborative filtering recommender system that makes personalized music recomendations. Users in this system describe their music taste by rating some songs. Each user has a profile that keeps track of all songs that the user has rated in the past, together with the ratings. All ratings are given on a scale from 1 to 7, with 7 having the highest value. Ringo compares user profiles to find users who have similar taste in music (they like or dislike same songs). Based upon preferences expressed by the most similar users, the system generates preferences for a given user. Specifically, each user can ask Ringo to (1) suggest new artist/album that the user will like, (2) list artists/albums that the user will dislike and (3) make predictions about a specific artist/album. When a user logs into Ringo for the first time,

the system sends a list of 125 artists and asks the user to give ratings for them. The user can chose not to rate artists he is not familiar with. The list is generated on a central server in two phases. In the first phase, the system choose the most popular artists from the database and adds them to the list. In the second phase, the system randomly selects some artists and adds them to the list as well. Besides recommendation services, this system offers some other services to the users. For example, each user can write short reviews about artists and albums, so other users can see and read those reviews.

2.2.4 Types of Algorithms for Collaborative Filtering

Collaborative filtering algorithms can be divided into two main categories - *Memory-based* and *Model-based* algorithms [8]. The first group of algorithms usually maintains tables for all user ratings and then perform some computations to get future predictions. The second group of algorithms first learn descriptive models of users and/or items and based upon that they generate recommendations. Model-based collaborative filtering algorithms are memory efficient, but time-consuming to build and update models. Memory-based algorithms are usually more accurate than model-based algorithms, but they are also more memory-consuming.

Memory-based Collaborative Filtering Algorithms

These algorithms utilize the entire user-item dataset to generate predictions. They are divided into two groups: *Item-based* and *User-based* methods.

User-based Algorithms These algorithms are predicting future ratings for a given user by finding a set of users known as nearest neighbors that have the most similar ratings to the target user.

Once when a set of the most similar users U_a for an active user a is found, the predicted rating for an item j for the active user a is computed as

$$p_{a,j} = \bar{v}_a + k \sum_{i \in U_a} w_{users}(a,i)(v_{i,j} - \bar{v}_i) \tag{2.1}$$

In the previous equation, $\bar{v}_a$ is the mean vote for the user a and k is a normalizing factor such that the absolute values of weights sum to unity. The mean vote is commonly used in these kind of computations to bring ratings from all users on a same level. Some users tend to be more generous in rating items, while other users tend to be more strict. The mean vote value is used to remove these differences in rating levels for different users. The weights $w_{users}(a,i)$ measures similarity between the active user a and each user in U_a. The next section explains common methods for computing w_{users}.

Computing User Similarities in User-based Collaborative Filtering Two most common approaches for computing similarities between pairs of users in user-based filtering are: *correlation-based similarity* and *cosine similarity*.

Correlation-based Similarity Pairwise correlation-based similarity between users in user-based collaborative filtering is computed by using the

Pearson-r correlation. This correlation measure can vary in magnitude from -1 to 1, with -1 indicating a perfect negative linear correlation, 1 indicating a perfect positive linear correlation and 0 indicating no linear correlation between two variables. The general formula for computing Pearson-r correlation between two variables X and Y that are given by a set of n measurements is

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2 \sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{2.2}$$

where

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i, \bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i \tag{2.3}$$

For computing Pearson-r correlation between two users a and b we first must isolate I_c, a set that contains all items that are rated by the both users. When we incorporate users data into the formula 2.2 (see [25]) we get the formula for computing correlation-based similarity as:

$$w_{users}(a,b) = \frac{\sum_{i \in I_c}(v_{a,i} - \bar{v_a})(v_{b,i} - \bar{v_b})}{\sqrt{\sum_{i \in I_c}(v_{a,i} - \bar{v_a})^2 \sum_{i \in I_c}(v_{b,i} - \bar{v_b})^2}} \tag{2.4}$$

where $\bar{v_a}$ and $\bar{v_b}$ are mean vote values for users a and b.

Cosine Similarity This similarity measure is incorporated into collaborative filtering algorithms from information retrieval. In information retrieval each document is represented as a vector with word frequencies, and similarities between two documents or queries and documents are com-

puted as cosine of the angle between two frequency vectors. In collaborative filtering, each user can be represented as a vector of item ratings. Similarity between two users is then computed as cosine of the angle between two rating vectors. For two given users a and b and their corresponding sets of rated items I_a and I_b respectively, the cosine similarity is expressed as:

$$w_{users}(a,b) = \sum_i \frac{v_{a,i} v_{b,i}}{\sqrt{\sum_{k \in I_a} v_{a,k}^2 \sum_{k \in I_b} v_{b,k}^2}} \tag{2.5}$$

Item-based Algorithms These algorithms are exploring item to item similarities to generate predictions for a given user. The item-based approach looks into the set of items that the target user has rated and computes how similar they are to a target item i, selects k the most similar items and based upon their ratings generates a prediction for the given user and the item i.

For a given user a and item j the prediction is computed as:

$$p(a,j) = \frac{\sum_{i \in I_{top_k}} w_{items}(i,j) v_{a,i}}{\sum_{i \in I_{top_k}} |w_{items}(i,j)|} \tag{2.6}$$

In the previous formula, $|w_{items}(i,j)|$ denotes the absolute value of $w_{items}(i,j)$. Methods for computing $w_{items}(i,j)$ are described in the next section.

Computing Item Similarities in Item-based Collaborative Filtering Three most common approaches for computing similarities between pairs of items in item-based collaborative filtering are: *Cosine similarity*, *Correlation-based similarity* and *Adjusted Cosine similarity*.

Cosine Similarity This similarity measure is very similar to cosine similarity measure in user-based collaborative filtering described earlier. For two items a and b and corresponding sets of users that these items U_a and U_b respectively, the cosine similarity is expressed as:

$$w_{items}(a,b) = \sum_i \frac{v_{i,a} v_{i,b}}{\sqrt{\sum_{k \in U_a} v_{k,a}^2 \sum_{k \in I_b} v_{k,b}^2}} \tag{2.7}$$

Correlation-based similarity For computing correlation-based similarity between two users, Pearson-r correlation is used, similar to computing correlation-based similarities in user-based approach that was described previously 2.8. If U_c is a set of users who rated both items a and b, the correlation-based similarity between these two items is expressed as:

$$w_{items}(a,b) = \frac{\sum_{i \in U_c} (v_{i,a} - \bar{r_a})(v_{i,b} - \bar{r_b})}{\sqrt{\sum_{i \in U_c} (v_{i,a} - \bar{r_a})^2 \sum_{i \in U_c} (v_{i,b} - \bar{r_b})^2}} \tag{2.8}$$

where $\bar{r_a}$ and $\bar{r_b}$ are mean vote values for items a and b.

Adjusted Cosine Similarity Computing pairwise item similarities by using cosine similarity as described in 2.5 has one important drawback. Unlike user-based collaborative filtering, where cosines of the angles between vectors are computed for each two pairs of users, in this approach cosines of the angles are computed between item vectors that contain ratings from different users. Therefore, the simple cosine similarity formula in item-based collaborative filtering is not taking into account the difference in rating scales

between different users. Some users tend to generally give higher ratings for all items, whereas some users tend to be more strict with their ratings. The adjusted cosine similarity offsets this drawback by subtracting the corresponding user average from each co-rated pairs of items. For given items a and b and set of users U_c that rated the both items, the adjusted cosine similarity is computed using the formula:

$$w_{items}(a,b) = \frac{\sum_{i \in U_c}(v_{i,a} - \bar{r_a})(v_{i,b} - \bar{r_b})}{\sqrt{\sum_{i \in U_c}(v_{i,a} - \bar{r_a})^2 - \sum_{i \in U_c}(v_{i,b} - \bar{r_b})^2}} \tag{2.9}$$

Model-based Collaborative Filtering Algorithms

In model-based collaborative filtering, a prediction for an item is computed using a probabilistic approach, as a expected value given the ratings of items that the user already rated. If we assume that a rating values are integer numbers between 0 and m then the expected rating for a given user a and a given item j is computed as:

$$p_{a,j} = E(v_{a,j}) = \sum_{i=0}^{m} Pr(v_{a,j} = i | v_{a,k}, k \in I_a) \tag{2.10}$$

where I_a is a set of all items that the user a has already rated. The next two sections describe two methods for model-based collaborative filtering - *Bayesian Classifier Model* and *Bayesian Network Model*.

Bayesian Classifier Model In the Bayesian classifier model [8], [21] the main idea is to separate users in classes. Rather than predicting the exact rating for a given item, this approach is trying to find which items to recommend and which not to. Each class is defined with certain set of preferences towards items. Given the class, the preferences towards the various items are independent. The probability that a user belongs to a class j, given his existing ratings (or feature values) is computed using the standard "naive" Bayes formula:

$$p(C = j|f_1, f_2, ..., f_n) = p(C = j) \prod_{i=1}^{n} p(f_i|C = j) \tag{2.11}$$

where both $p(C = j)$ and $p(f_i|C = j)$ are estimated from training set containing user ratings. To determine to which class the given user belongs to, the probability of each class is computed and the user is assigned to the class with the highest probability. Although the assumption that preferences are independent given the class is sometimes unrealistic, this approach proved to be very accurate in predicting new items, even in situations where this assumption does not hold.

Bayesian Network Model In the Bayesian network model [8] entire domain is represented as a Bayesian network with each node representing each item in the domain. The states of each node correspond to the possible rating values for each item. The "no vote" state is also included, where

there is no interpretation for missing data. Each node has a decision tree. Conditional probabilities for each node (item) are represented as a decision tree. After the learning process is completed each item will have a set of parent items that are the best predictors for a given node.

2.3 Hybrid Filtering

This type of information filtering combines resources of the both previously mentioned approaches. The main objective of this approach is to overcome some limitations imposed by either of the approaches. The following are some systems that are filtering information using this approach.

FAB System The FAB system [3] is an adaptive web pages recommendation service system that seeks to adapt to its users. It is a distributed implementation of a hybrid system and it was developed at the Stanford University. The Fab system consists of three parts – selection agents, collection agents and central repository. Documents in this system as well as users' profiles are represented as vectors of their words using the Vector Space Model (VSM) (see [26]). Each user is accompanied with a selection agent that maintains the user's profile. When a user requests a recommendation for web pages, its selection agent sends a request to one of the existing collection agents. The collection agent then tries to find top-k web pages in the central repository that have the highest matching scores with the user's profile. The

matching function is a variant of the standard information retrieval cosine measure between the vectors of pages and the user's profile vector. After the top-k web pages are presented to the user, he rates each page on the scale from one to seven. After the user rated web pages, the selection agent updates the user's profile. The request-retrieval processes in this system use methods and approaches from content-based information filtering. The collaborative filtering part of this system is implemented through a feature that sends web pages that are highly rated by a user to its nearest neighbors. Nearest neighbors of a user are defined as users that have profiles that are most similar to the given user.

This thesis proposes novel approaches for filtering information using collaborative filtering methods. Like other collaborative filtering systems that are described in this chapter (Ringo, MovieLens, GroupLens), our proposed CF models and methodologies are using ratings given by a set of users to filter relevant information. However, unlike the previously mentioned systems, our proposed approaches are different in the terms that they are neither purely user-based nor item-based methods, but synergies of two. We are also using novel mathematical models to represent ratings data from datasets, that are, unlike approaches in these models, closely related to graph theory approaches.

Chapter 3

Collaborative Items Filtering With Maximum User Satisfaction - Models and Heuristic Approaches

In this thesis a new framework, called *collaborative partitioning*, or CP for short, is proposed. CP focuses on the problem of identifying item partitions that optimize the fraction of users who will be satisfied with a given partition. In contrast to traditional clustering, the proposed CP model is a novel, bi-criteria optimization problem, which has a goal of finding the best clustering that satisfies the most users at the highest level of satisfaction. The CP framework proposed in this thesis considers two natural models for

determining user satisfaction level with a given single partition.

The motivation for studying the collaborative partitioning problem comes from analyzing some recommendation systems and working and analyzing large data sets with preference data. StumbleUpon [29], one of the largest social media websites on the Web nowadays is used to direct people to recommend websites. As it was mentioned before, users can state whether they like or dislike certain web pages by giving them "thumb-up" or "thumb-down" ratings, or they can give no ratings to pages which means that they have no opinion about a certain webpage. In many cases in this recommendation system, topics that are recommended to a user sometimes are not a very good match of the user's interest. This problem indicates that topic hierarchies have to be managed to better meet user needs, allowing them to more narrowly define their topics interests. Collaborative partitioning would be very helpful in finding future interesting topics for a user, based on his ratings history.

The second motivating example of a motivation for this work grows out from the experiments that we performed on the Netflix Prize database [23] of user preference opinions about movies. In attempting to organize the set of movies into segments and partitions we found that correlation values on pairs of films in same (human-defined) categories were very widely distributed, and in addition very little transitivity in correlation values was exhibited among movies in same category. For example, we considered the correlations among

ratings of movies categorized by one website as "Top 10 Word War II Films" [16]. This set consisted of the following movies: 1. Saving Private Ryan, 2.Pearl Harbor, 3. Schindler's List, 4.The English Patient, 5.Life is Beautiful, 6.U571, 7.The Pianist, 8.Enemy at the Gates, 9.The Thin Red Line, 10.Patton. Through ad-hoc experimentation we found that a partitioning of these items into {1,3,4,5,7,10} and {2,6,8,9} exhibited very high average correlation values when measured over a large subpopulation of users. In this thesis we are trying to focus on approaches of generating this kind of partition automatically, by considering the application of expressed preference data in the context of partitioning for the purpose of satisfying the most users. This chapter introduces a novel formal framework for collaborative partitioning, algorithms and analysis that demonstrate the potential and limits for solving these problems. We propose two models for collaborative partitioning, analyze their computational complexities and propose heuristic algorithmic approaches for these two models. At the end of this chapter, we demonstrate experimental results with real data for these two models.

3.1 Related Work

The CP model is related to the problem of weighted graph clustering (see [7]). A specific type of graph clustering, called correlation clustering, was introduced in [4]. Correlation clustering considers a graph with edges labeled from set $\{+, -\}$, and considers the problem of clustering to either maximize

agreements, the number of + edges within clusters, plus the number of − edges between clusters (or equivalently, minimizing the number of disagreements). Similar to the approach in [4], there are several other papers that are trying to improve constant-factor approximation algorithms for maximizing agreements (see, [30, 11]).

However, the problem of clustering using a *correlation measure*, that is, maximizing the sum of agreements minus disagreements appears to be more difficult problem to approximate. Correlation clustering using a correlation measure may be seen as comparable to our Collaborative Partitioning CP problem, since we seek to measure the quality of a partitioning in terms of a distance measure that involves the (sum of) positive and negative satisfaction of individual users. In addition, our CP model differs from the standard graph clustering and correlation clustering, in the sense that our goal is a bicriteria optimization in which we seek the partitions that satisfy a maximum number of users at the highest level. The CP model can also be compared to biclustering, in which both users and items are clustered simultaneously. However, traditional biclustering [31], [32] in collaborative filtering applications have tended to focus only on positive preference data and ignore negative preferences.

3.2 Mathematical Notation and Formal Framework for Collaborative Partitioning Models

Collaborative partitioning (CP) model of preference data consists matrix $M = [a_{ij}]$, where $a_{i,j} \in \{-1, 0, 1\}$, and rows are indexed by n users U and columns are indexed by m items I. Typically we have $m < n$. Here $a_{ij} = +1$ indicates a positive opinion by user i on item j, $a_{ij} = -1$ indicates negative opinion, and $a_{ij} = 0$ indicates no opinion. The restriction we use simplifies the presentation, and many of our results generalize to arbitrary values in $[-1, 1]$. Also, in practice, the matrix M is chosen as a submatrix of a larger preference data set.

In this work our goal is to partition the itemset I into two or more (non-overlapping) segments $P(I) = \{I_1, I_2, \ldots, I_k\}$ such that a large fraction of users are highly satisfied with one of the subsets I_i. The idea is that the segment I_i can stand as a representative of user preferences. In typical applications, users could select the segment of the partition best meeting their needs, or the segment of the partition could be determined automatically. The overall goal of the partition would be to improve system performance and user satisfaction. We now consider two models for quantifying the quality of given partitions. The first or A-model is focussed on creating partitions that maximize the satisfaction of the most individual users. The second or

B-model is focussed on creating partitions of items that are similar and well correlated with respect to the preferences of a large fraction of users.

3.3 Two Models For Modeling User Satisfaction

3.3.1 A-model of User Satisfaction

The A-model for CP measures the quality of partitions based on the number of threshold-satisfied users. Each user's preference data (stored as a row in matrix M) is represented by a vector $\hat{u}$ of length m. For a given partition $P(I) = \{I_1, I_2, \ldots, I_k\}$, we let $\hat{I}_j$ be the vector of length m over ± 1 representing the segment I_j, i.e., $\hat{I}_j(i) = 1$ if $i \in I_j$ and -1 otherwise. The dot product $\hat{u} \cdot \hat{I}_j$ yields a count of the number of positive u-preferences minus the number of negative u-preferences in I_j plus the number of negative u-preferences minus the positive u-preferences in complement $I_j^c = I - I_j$. We say a partition $P(I)$ of the itemset *β-satisfies* u if this dot product $\hat{u} \cdot \hat{I}_j \geq \beta ||\hat{u}||^2$, for some $1 \leq j \leq k$. Here we take the Euclidean norm $||u|| = \sqrt{\sum u_i^2}$. The parameter β is used to represent the threshold specifying the required level of user satisfaction, and represents the fraction of the user's preferences that are consistent with the partition.

Definition. An itemset partition $P(I) = \{I_1, I_2, \ldots, I_k\}$ induces (α, β)-satisfaction in the A-model (for some $0 < \alpha, \beta \leq 1$) if $P(I)$ β-satisfies at

least $\alpha|U|$ users.

With this definition we have the associated bicriteria optimization problem, which seeks to find the partition that optimizes (α, β)-satisfaction. In the A-model the number of segments/clusters k is not an issue for the bicriteria optimization and will be selected automatically.

3.3.2 B-model of User Satisfaction

In this model the quality of partitions is measured by focusing on correlation clusterings of items, which uses a benefit-measure based on items similarity (or correlation) exhibited by each cluster with respect to large numbers of user preferences. The B-model uses graph clustering and a natural, parameterized cost model. We consider the family of weighted graphs $\mathcal{G} = \{G_{U'}\}$ on the same set nodes representing the itemset I; and for each subset of users $U' \subseteq U$ there is a graph $G_{U'}$ in the family where weights on the edges are given by correlation values between items, where the correlations are calculated by restricting to U' data. In general, for the B-model, it is reasonable to choose any vector similarity measure, but we focus on cosine similarity for sake of concreteness. Given two column vectors c_i, c_j of M, and a subset U', we let $sim_{U'}(i, j)$ be the cosine similarity measure of the two column vectors restricted to the rows of U'. Hence, for the graph $G_{U'}$ the edge (i, j) has weight $sim_{U'}(i, j)$.

We use a natural cost model for the partitioning problem on each $G_{U'}$, where there is a positive benefit equal to the weight of a positive edge if it lies

within a cluster, and induces a negative benefit otherwise, and vice versa for negatively weighted edges. The objective here is to find a subset U' of sufficiently large size, such that the graph $G_{U'}$ can be partitioned with maximum benefit - maximizing the positive correlation among items within clusters and negative correlation among items between clusters, and minimizing the reverse.

Definition. An itemset partition $P(I) = \{I_1, I_2, \ldots, I_k\}$ induces (α, β)-satisfaction in the B-model (for some $0 < \alpha, \beta \leq 1$) if there exist a set $U' \subset U$ of rows where $|U'| \geq \alpha|U|$ such that the weighted graph $G_{U'}$ when partitioned according to $P(I)$ has total benefit at least $\beta\binom{|I|}{2}$, i.e., the average benefit per edge is at least β.

For sufficiently large values α, β, a partition P that induces (α, β)-satisfation in the B-model identifies a large subpopulation of users for which the pairwise similarity values of items are consistent (on average) with the partitioning. Like the A-model, in this B-model of CP the number of segment/clusters k is an not an important consideration, and will be chosen automatically.

3.4 Computational Complexity

This section discusses computational complexity of CP in A-model and B-model. It can be proved that the decision problems associated with both A and B models of CP are NP-complete. The proofs are based on showing that solutions for the both problems are verifiable in polynomial time, as

well as the optimization problems for the both models are NP-hard. That can be proved by showing that the CP problem for (α, β)-satisfaction in the A-model and the B-model are different generalizations of the correlation clustering problem, which is shown to be NP-complete in [4].

Theorem 3.1. *The CP problems of optimizing* (α, β) *- satisfaction in the A-model and optimizing* (α, β)*-satisfaction in the B-model are both NP-Hard. The problems remain so for the restricted case where* $\beta = 1$ *in the A-model, and for the restricted case where* $\alpha = 1$ *in the B-model.*

Proof. The NP-hardness of optimizing (α, β)-satisfaction in the A-model and B-model can be proved by reducing the correlation clustering problem to CP partitioning in both the A- and B- models.

For reducing the correlation clustering problem to the CP problem in the A model, we take an arbitrary instance of a correlation clustering problem, which is a graph with edges labeled with $\{+, -\}$. This input graph is transformed to an instance of an A-model CP problem by applying the next reduction. We represent the input graph by a matrix M that has rows indexed by the edges and columns indexed by the nodes of the input graph. For each edge labeled with $+$, we create a row in the matrix M with both associated columns set to $+1$, and all other values set to 0. For each edge labeled with $-$, we create a row in the matrix M with labeling one of the associated columns with $+1$, the other with -1, and setting all other values to 0.

For any partition of the columns, we can see that a row in the M matrix

is 1-satisfied if and only if the associated edge is in agreement with a given partition. If f is a number of edges in M that are 1-satisfied with a given partition, then $\alpha = f/|E|$ fraction of edges are in the agreement with a given partition if and only if the partition yields $(\alpha, 1)$-satisfaction in the A-model. The correlation clustering problem can be reduced to CP partitioning in the B-model by using the same reduction. In a previously constructed matrix M it holds that a cosine between any pair of columns is 1 if and only if the edge associated with the corresponding pair of nodes is labeled $+$, and it is -1 otherwise. Therefore, the $^{\pm}1$- weighted sums of the dot products over all pairs on columns is the sum of the agreements minus disagreements in the correlation clustering graph. So the correlation clustering reduces to the problem of finding beta-maximizing $(1, \beta)$-satisfaction in the B-model. $\square$

The next theorem is discussing a computational complexity of finding subsets of users that yield high correlation values in CP.

Theorem 3.2. *The problem of finding a subset $U' \subset U$ of rows where $|U'| \geq \alpha|U|$ such that the weighted graph $G_{U'}$ (as defined above) edge weights at least β is NP-Hard.*

Proof. To prove this theorem, we take an arbitrary instance of the k-Independent Set problem with $G = (V, E)$, $n = |V|$ and $m = |E|$. Without loss of generality we can assume that $k = \frac{n}{2} > 3$. We transform this input into a $n \times (m+n)$ matrix M that has rows indexed by the nodes in V and columns indexed by nodes and edges in G. for each entry of M, we set $M_{ij} = -1$ if the node $i \in j$,

in a case when j is an edge, or $i = j$, in a case where j is a node. Otherwise, we set $M_{ij} = 1$. The claim is that G has a k-Independent Set if and only if the matrix M has k rows such that the minimum pairwise cosine similarity is at least $\frac{(k-4)}{k} = 1 - \frac{4}{k}$.

We can observe that, if G has a k-Independent Set, then the rows in M corresponding to the nodes in IS in G have a property that each column vector reduced to these rows in M has at most one -1 entry. Therefore, any pair of columns reduced to the previously mentioned row set has at most two row indices with opposite values. So the cosine similarity between each two pairs of columns is at least $\frac{\sum_{i=i}^{(k-2)} 1*1+1*(-1)+1*(-1))}{k} = \frac{(k-4)}{k}$.

Conversely, assume that M has k rows that have the minimum pairwise cosine similarity is at least $\frac{(k-4)}{k}$. Assume also that this set does not correspond to a k-Independent Set in G, $i.e$ there are two rows in this set that correspond to nodes u and v that share an edge e. The column for the edge e in M reduced to the previously described k rows has two -1 values. Therefore, the cosine similarity between this column and any other column that corresponds to w, $w \neq u, v$ has value at least $\frac{\sum_{i=1}^{(k-4)} 1*1+1*(-1)+1*(-1)+1*(-1)+1*(-1)}{k} = \frac{(k-6)}{k}$ which is in the contradiction with the starting assumption.

□

3.5 Heuristic Approximations for Solving CP Problems

In the section 3.4 it was shown that, in general, CP problems are NP-hard. Therefore, CP applied on large sized problems should be considered intractable and not scalable. To be able to find approximate solutions for our proposed CP models, we examined some heuristic approaches that give approximate solutions in polynomial time. This section gives details about our proposed heuristic approximations for CP problems.

In our proposed heuristic approach for approximating CP solutions, we start with a polynomial size family $\mathcal{P}$ of partitions that can potentially yield approximately optimal results. Many clustering methods on weighted graphs are available, and it is not clear at this point which method or methods are to be preferred; hence, for our purposes we try all reasonable methods based on similarity clustering, and attempt to determine which is optimal or nearly so.

Once given a family of partitions $\mathcal{P}$ we proceed as follows. For an instance of the A-model CP problem, we will simply count the number of rows that are β-satisfied for each partition in the family of partitions $\mathcal{P}$, and then output the partition that yields the best result.

For an instance of the B-model CP problem we have a more complex problem to solve since deciding which rows to use in calculating the satisfaction is not immediate, as they are not independent decisions (previous

decisions affect all future decisions). We chose to take a greedy approach to deciding which set of rows to use. We begin with all rows and then remove the row that best improves the overall cost benefit. We iteratively repeat this greedy process until no such row can be found, or if the α threshold has been reached.

3.5.1 Methods for Generating a Family of Partitions

In our experiments we considered the generation of a family of partitions given by the approaches in *Hierarchical Agglomerative Clustering* (HAC). These agglomerative algorithms are used frequently in practice, and can easily and efficiently create families of partitions. The motivation behind using HAC methods to create families of partitions is that these methods hierarchically and gradually create clusters of items, based upon their similarity, and our intuition is that one partition among created families of partitions can be a very good starting point in our heuristic approach. We considered six different HAC methods in our experiments:Single Linkage Method, Average Linkage Method, Complete Linkage Method, Weighted Average Linkage Method, Centroid Method and Median Method. We chose these six methods because these are all existing HAC methods that are agreeable with the cosine similarity measure that is used in the clustering process. HAC algorithm works as follows (see 3.1): It finds the closest two segments I_i and I_j, it merges them and then it updates distances

Algorithm 3.1 Basic agglomerative clustering algorithm for generating family of partitions

1: INPUT: $n \times m$-matrix of preference data M
2: OUTPUT: A family of partitions $\mathcal{P}$ of the columns
3: Let $P = [\{1\}, \{2\}, \ldots, \{m\}]$ be the finest partition containing m cluster segments
4: Add copy of P to $\mathcal{P}$
5: Compute the $m \times m$ inter-cluster distance matrix $\mathcal{D}$ by using cosine similarity measure for each pair of columns in M
6: **while** P has more than one segment **do**
7: Find closest pair of segments I_i, I_j in P
8: Merge I_i and I_j into one segment in P
9: Add P to $\mathcal{P}$
10: Compute the distance between the new cluster segment $I_i + I_j$ and all other segments
11: Add the new row for $I_i + I_j$ and delete the rows and columns associated with I_i, I_j from $\mathcal{D}$
12: **end while**
13: Return family of partitions $\mathcal{P}$

between a new segment $I_i + I_j$ and any other remaining segment. The distance is computed using four weight parameters: $(\delta_1, \delta_2, \delta_3, \delta_4)$ as follows: $d(I_i + I_j, S) = \delta_1 d(I_i, S) + \delta_2 d(I_j, S) + \delta_3 d(I_i, I_j) + \delta_4 |d(I_i, S) - d(I_j, S)|$. For different HAC methods, the weight parameters are computed differently:

- Single Linkage Method uses $(1/2, 1/2, 0, -1/2)$
- Complete Linkage Method uses $(1/2, 1/2, 0, 1/2)$
- Average Linkage Method uses $(1/2, 1/2, 0, 0)$
- Weighted Average Linkage Method uses $(\frac{|I_i|}{|I_i|+|I_j|}, \frac{|I_j|}{|I_i|+|I_j|}, 0, 0)$
- Centroid method uses $(\frac{|I_i|}{|I_i|+|I_j|}, \frac{|I_j|}{|I_i|+|I_j|}, -\frac{|I_i||I_j|}{|I_i|+|I_j|}, -\frac{|I_i||I_j|}{(|I_i|+|I_j|)^2})$

- Median method uses $(1/2, 1/2, -1/4, 0)$

Each HAC method generates $m - 1$ partitions of an m-item set, as described in 3.1. By using a priority queue to extract the closest pair, the time total complexity is therefore $O(m^2 \log m)$ in the worst case. This follows since updating the distance matrix takes $O(1)$ time for each remaining segment. However, in some cases it may be possible to use a more effective data structure to reduce the time complexity.

3.5.2 A-model Algorithm

The running time for the A-model algorithm is bounded by the time it takes to generate the family of partitions $\mathcal{P}$ plus the time it takes to process each of the partitions. For each of the partitions under consideration we process each row to determine whether it is β-satisfied. This function takes time $O(m)$. Hence, assuming, as above, that the partitions can be generated in $O(m^2 \log m)$, the total time complexity is $O(m^2 \log m + nm^2)$.

3.5.3 B-model Algorithm

Similar to the algorithm for the A-model, the running time for the B-model algorithm is bounded by the time it takes to generate the family of partitions $\mathcal{P}$ plus the time it takes to process each of the partitions in the family. However, the B-model algorithm is little more complicated, for each of the

partitions under consideration we greedily determine the set of rows over which to calculate the cost benefit of the partition. In each iteration of this greedy operation we must consider each row and compute the cost benefit induced by the removal of the row. Once the partition that maximizes the benefit for a given α (over all partitions) is computed, the β satisfaction can be computed using the definition for the (α, β)-satisfaction in the B-model as $MaxBenefit = \beta\binom{|I|}{2}$, where I is the set of items we are partitioning.

It easily follows that to process each partition the algorithm above takes time $O(nm^2)$. Hence, assuming, as above, that the partitions can be generated in $O(m^2 \log m)$, the total time complexity is $O(m^2 \log m + nm^3)$.

3.6 Applying Collaborative Partitioning Method to Produce Top-K Item Recommendations

In this section we propose approaches for applying our proposed CP frameworks for building $top-k$ recommendation systems. The $top-k$ recommendation algorithms in collaborative filtering are designed to answer $top-k$ queries. For a given user a and, given set of items I_a and a given non-negative number k, the $top-k$ recommendation algorithms are trying to find a set of up to k items that have not yet been rated by the user a, but are most likely to receive the highest rankings and to be highly preferred by the given user among all items that the user has not yet rated. The order

of the items reflects the probability of the user's preference towards these items – the higher position an item has in a $top-k$ recommendation list, it is more likely to be preferred compared to some other items that are ranked lower in the same list. Our framework for applying the CP approaches for $top-k$ recommendation purposes is still in the developing phase. So far, we designed and evaluated our $top-k$ recommendation algorithm based on the CP approaches proposed in the B-model that was described in 3.3.2.
Our $top-k$ recommendation algorithm takes clusters of users produced by the B-model algorithm given in last section.

The input of our $top-k$ recommendation algorithm is the training user-ratings matrix, the testing user-ratings matrix, minimal percentage of targeted users min_u, the minimal users satisfaction that has to be achieved in the produced clusters min_sat, and the number of the nearest user. The training user-ratings matrix is a matrix that contains incomplete user ratings, *i.e.*, some of the ratings are known but removed from this matrix for the purpose of testing the accuracy of the produced recommendations. The testing user-ratings matrix contains complete data and is used for evaluating accuracy of our $top-k$ recommendation approach. We iteratively apply the CP algorithm for the B-model as long as this algorithm produces new clusters with the β satisfaction above the threshold satisfaction value min_{sat} and as long as there are any users left to be clustered, as described in the Algorithm 3.4. After we obtained the clustering for the training data, we produce $top-k$ recommendations for the users and missing items and compare the results

with the testing data. We produce the $top - k$ recommendation list for each user a and set of the user's missing items I_{ma}, by allocating the cluster the current user belongs in, finding s nearest users to the given user (using the cosine similarity on the vectors of users), and then, for each missing item i of the user a we compute sum of ratings given for that item by the user's nearest neighbors. After we apply these steps for all missing items for the user a, we order items by the computed sums in descending order, and recommend the $top - k$ items with the positive rating sums. Detailed steps of the recommendation algorithm are given in the Algorithm 3.5.

The clusters produced by the CP methods are able to speed up the recommendation process significantly, since they group similar users into same clusters. Therefore, when we are searching for nearest neighbors for a given user in our recommendation algorithm described in 3.5, we are searching only through the users that are located in the same clusters with the given user.

Algorithm 3.2 Algorithm for maximum user satisfaction in the A-model of CP

INPUT: $n \times m$-matrix of preference data M; the threshold value $\beta > 0$.
OUTPUT: A partition $P \in \mathcal{P}$ to optimize the number of rows that are β-satisfied.
Generate a family $\mathcal{P}$ of partitions of the m columns. (As illustrated in Algorithm 1.)
Let *BestPartition= []*
Let *MaxSatCount=0*
for all partitions $P = \{I_1, I_2, \ldots, I_k\} \in \mathcal{P}$ **do**
  Pcount=0 // counts number of β-satisfied users
  for all row vectors $\hat{u}$ of M **do**
    MaxSat[u] = 0
    for all $1 \leq j \leq k$ **do**
      // Find the segment-vector $\hat{I}_j$ that maximizes satisfaction of u
      Calculate the dot product $p = \hat{u} \cdot \hat{I}_j$
      if ($p >$*MaxSat[u]*) **then**
        MaxSat[u] = p
      end if
    end for
    if (*MaxSat[u]*$> \beta$) **then**
      Pcount = Pcount +1
    end if
  end for
  if (*Pcount* $>$ *MaxSatCount*) **then**
    MaxSatCount =Pcount
    BestPartition = P
  end if
end for
Return *BestPartition*

Algorithm 3.3 Algorithm for maximum user satisfaction in the B-model of CP

INPUT: $n \times m$-matrix of preference data M; the threshold value $\alpha > 0$
OUTPUT: A Partition $P \in \mathcal{P}$, so as to optimize the β value, so that P is an (α, β)-satisfying partition.
Generate a family $\mathcal{P}$ of partitions of the m columns. (As illustrated in Algorithm 1.)
Let *BestPartition= []*
Let *MaxBenefit=0*
for all partitions $P = \{I_1, I_2, \ldots, I_k\} \in \mathcal{P}$ **do**
  for all pairs of columns $i < j$ **do**
    calculate w_{ij} = cosine similarity of columns i and j of M
    if pair i, j are in same segment I_s of P **then**
      w_{ij} is in set of *Inter-Cluster-Weights*
    else
      w_{ij} is in set of *Intra-Cluster-Weights*
    end if
  end for
  P-CostBenefit= Σ *Intra-Cluster-Weights* - Σ *Inter-Cluster-Weights*
  while there remains at least αn rows of M and the selected maximum *P-BenefitChange[u]* > 0 **do**
    for all rows u of M **do**
      determine the *P-BenefitChange[u]* that incurs when the sums of *Intra-Cluster-Weights* and *Intra-Cluster-Weights* change resulting from removing u from M
    end for
    Greedily select and remove row u from M for which *P-BenefitChange[u]* is maximized
    if (the selected maximum *P-BenefitChange[u]* > 0) **then**
      Add *P-BenefitChange[u]* to *P-TotalBenefit*
    end if
  end while
  if (*P-TotalBenefit* $>$ *MaxBenefit*) **then**
    MaxBenefit = P-TotalBenefit
    BestPartition=P
  end if
end for
Return *BestPartition*

Algorithm 3.4 User clustering algorithm based on the B-model CP

1: INPUT: $n \times m$-matrix of preference data M; the threshold satisfaction value $min_{sat} > 0$; the minimal percentage of the targeted users min_u;method m of HAC;
2: OUTPUT: Clusters of users with similar agreements on CP items $C_1, ...C_m$
3: $Users_{unclustered} = M$;
4: $Clusters_{counter}$=0;
5: **while** there remain unclustered users and produced β-satisfaction $Sat_{current} > min_{sat}$ **do**
6: Generate family of partitions P= $HAC(Users_{unclustered})$
7: $Percent_{current} = min_u$
8: $(Users_{satisfied}, Sat_{current}) = Model_B(Users_{unclustered}, P, Percent_{current} \cdot n)$
9: **while** $Sat_{current} < min_{sat}$ **do**
10: $Percent_{current}$=$Percent_{current} - 1$
11: $(Users_{satisfied}, Sat_{current}) = Model_A(Users_{unclustered}, P, Percent_{current} \cdot n)$
12: **end while**
13: $Clusters_{counter} = Clusters_{counter} + 1$
14: $C[Clusters_{counter}] = Users_{satisfied}$
15: $Users_{unclustered} = Users_{unclustered} \bigcap Users_{satisfied}$
16: **end while**
17: Return *clusters* C

Algorithm 3.5 Top-k recommendation algorithm based on clustering produced by the B-model

INPUT: Clusters of users with similar agreements on CP items $C_1, ...C_m$ from algorithm 3.4;sets I_1,I_2,...,I_k that contain missing items ratings for users U_1,...U_k;value k;number s of nearest users
OUTPUT: for each user U_i from the input, the $top-k$ recommendation items list of the length k
for all users U_i **do**
  Let $recommend_i$={}
  for all items i_j in I_i **do**
    $vote_j = 0$
    for all nearest neighbors U_{ni} in the set of s nearest neighbors for U_i **do**
      $vote_j = vote_j + rating(U_{ni}, i_j)$
    end for
    Add $vote_j$ in $recommend_i$
  end for
  Sort items in $recommend_i$ in descending order
  Recommend $top-k$ or less items from $recommend_i$ that have a positive score
end for
Return *collection of recommend sets*

Chapter 4

Collaborative Items Filtering With Maximum User Satisfaction - Example Results with a Real Dataset

In this chapter we present the results that we obtained after running proposed heuristic approaches for the A- and B-model and the $top-k$ recommendation approach on real data. By applying the previously mentioned approaches on real data, we wanted to test the next features:

- How changes of one parameter (α or β) affect the other one, in the models A and B. We would like to see, for the particular dataset we

are experimenting with, what are the maximal α-fractions of β-satisfied users for given values of β (in the A model), and for given desired α-fractions of users, what are maximum β-satisfaction values that we can achieve (in the B-model).

- How much the best partitions computed using algorithms for the A and B models are similar to a human-generated genre partition of given items. In this particular experiment, our dataset consists of users preference data for a given set of movies from different genres. So we would like to see how far are the best partitions obtained from the algorithms for the A and B models from a genre partition, where movies (items) are classified according to their genres. Our assumptions are that the proposed heuristic algorithms are able to remove noise from the dataset and to create partitions that are relatively close to the genre partition.

- How the various proposed HAC methods perform for a particular dataset we are experimenting with.

- Examine whether heuristic approaches for the A and B model are finding significant and meaningful item partitions (if they exist). So we would like to test what kind of partitions our proposed algorithms are finding on randomly generated dataset and compare these results against results obtained on a real dataset.

- We would like to evaluate the quality of $top - k$ recommendations cre-

ated with our proposed recommendation algorithm and how they compare in contrast with some other recommendation algorithms.

For our experimental setup, we chose a subset of the recently released Netflix movie reviews. This set contains around 100000000 ratings for movies, on the scale from 1 to 5. Because in our approach we are dealing with more simple, {1,-1,0} ratings, we translated the ratings from the 5 scale to positive and negative ratings. From this dataset, we chose a subset that contains ratings for randomly chosen 23 movies that belong to three different genres – movies about James Bond 007, movies that are directed by Woody Allen and black and white movies. The chosen subset contains ratings from 1429 users, who rated at least 15 out of 23 randomly chosen movies. Movies that are chosen belong to the three major topics – black and white movies, movies directed by Woody Allen and movies about James Bond, the 007 agent. The titles of the 23 movies that we chose are given in the table 4. Each movie belongs to one of the previously mentioned, human-defined topics:movies either written or directed by Woody Allen {1,2,3,4,5,6,7,8}, black and white movies {17,18,19,20,21,22,23} and movies about 007 agent, James Bond {9,10,11,12,13,14,15,16}. Some of the movies in the first group can be also put in the second group, because they are also black and white moves. From this dataset we created preference partitioning P by applying the algorithms described in the A-model and B-model on a family of initial partitions produced by six different Hierarchical Agglomerative Clustering methods - Single Link, Average Link, Weighted Average Link, Complete Link, Median

and Centroid. We ran two sets of experiments for each model. One set of experiments measured α-percentage of β-satisfied users in the both models, for partitioning produced by six different HAC methods. The second set of experiments measured *Jaccard error distance* between produced partitioning in each model and the human-based genre partitioning that was listed in the table in 4. *Jaccard error distance* between two item partitions P_a and P_b is defined as:

$$Error_{Jaccard}(P_a, P_b) = \frac{\sum_{I_i \in P_a} Min_{I_j \in P_b}(1 - \frac{|I_i \cap I_j|}{|I_i \cup I_j|})}{|P_a|} \tag{4.1}$$

We can see from the formula 4.1 that, if we replace P_a with a human-generated partition and P_b with any produced partition that *Jaccard error distance* is simply the average Jaccard distance of each of the segments in the human-generated partition from optimally chosen segment in the given partition.

We also ran the same sets of experiments for randomly generated data, to be able to determine whether our methods are finding statistically relevant segments and partitioning.

Table 4.1: Human-based genre partitioning of the experimental itemset

Woody Allen Movies
1."Love and Death", 2."Mighty Aphrodite", 3."The Curse of the Jade Scorpion", 4."Annie Hall", 5."Deconstructing Harry", 6."Hannah and Her Sisters", 7."Crimes and Misdemeanors", 8."Manhattan",
Movies about James Bond
9."GoldenEye", 10."Tomorrow Never Dies", 11."Die Another day", 12."The Spy Who Loved Me", 13."The World is Not Enough", 14."Licence to Kill", 15."The Living Daylights", 16."For Your Eyes Only"
Black and White Movies
17."Lenny", 18."The Last Picture Show", 19."Paper Moon", 20."Psycho", 21."The Best Years of Our Lives", 22."Some Like It Hot", 23."From Here To Eternity"

4.1 Results for the A-model

In running experiments for the A-model, we followed steps of the algorithm described in the Algorithm 3.2. Input value for the algorithm was β parameter, and then we measured percentages of the β-satisfied users and Jaccard error distance from the genre partitioning for families of partitions produced by each of the six different hierarchical agglomerative clustering methods listed above. The Figure 4.1 describes experimental results for (α, β)-satisfaction in the A-model. As we can see from the Table 4.1 and the Figure 4.1 the best partition for all α and β values is partition {9, 10, 11, 12, 13 14, 15, 16} {3} {1, 2, 4, 5, 6, 7, 8, 17, 18, 19, 20, 21, 22, 23}. This partitioning is produced by applying the A-model algorithm on the partitions produced by Single Link and Centroid HAC. If we look at different segments in the parti-

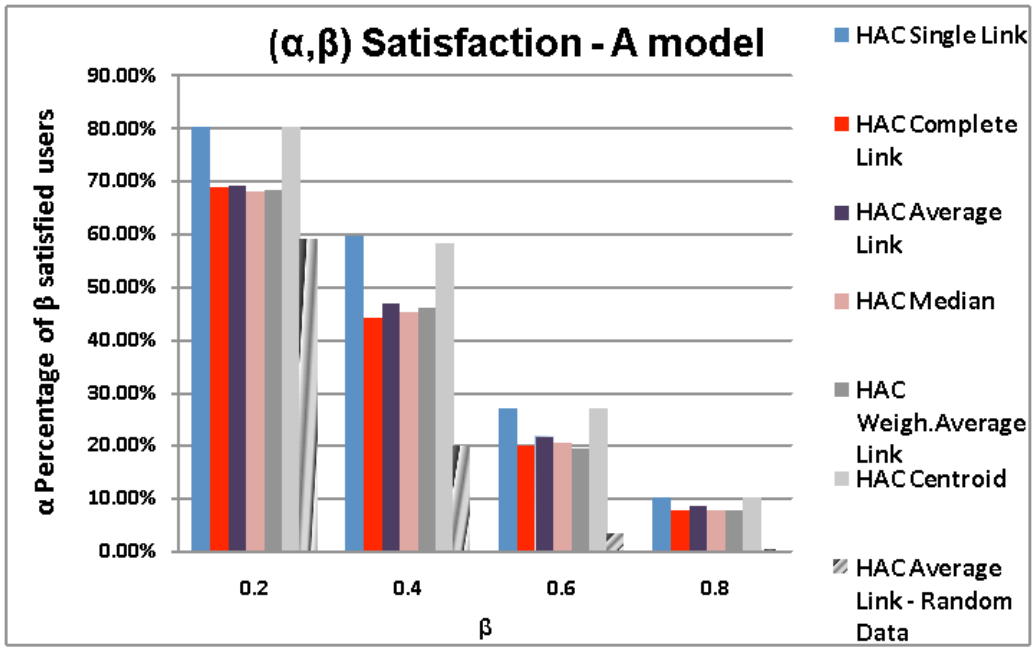

Figure 4.1: (α, β)-satisfaction results in A-model

tion, we can see that all movies about James Bond are in the same segment. That means that, in our data set, majority of the satisfied users agreed on their preference opinion about these movies. The second large segment in this partition is the segment { 1, 2, 4, 5, 6, 7, 8, 17, 18, 19, 20, 21, 22, 23}. This segment contains together all black and white movies and all but except one movie from Woody Allen movies group. From that segment we can conclude that for the majority of satisfied users agree about their preference opinion about these movies (they like or dislike them in the same way). If we analyze the Figure 4.1, we can notice that partitions produced by all HAC methods follow a same trend – the higher β-satisfaction requirement will produce smaller clusters. For β value of 0.8 we can see from the figure that only

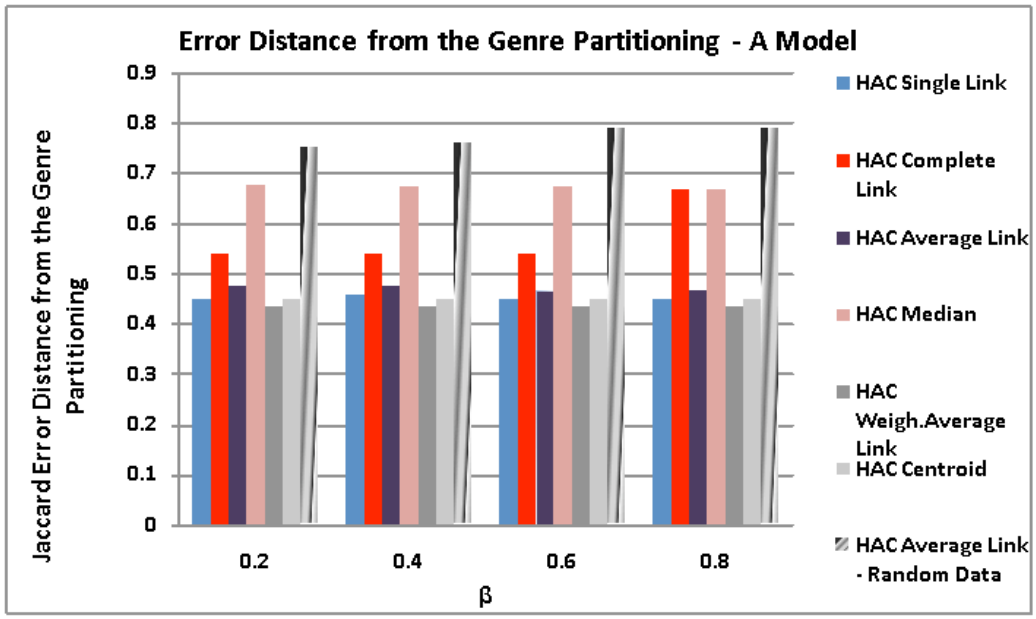

Figure 4.2: Measured error distance between produced preference partitions and the genre partitioning for the A-model

10 percent or less users are going to be 0.8 satisfied with a given partition. However, users in those types of clusters tend to be much more similar in taste to each other than users in clusters that are only 0.2 or 0.4 satisfied. In the Figure 4.1 we also included experimental results that we obtained by applying our algorithm on HAC partitions applied on randomly generated data. We included only results produced by applying the A-model algorithm on partitions produced by applying Average Link HAC on the random data, since that particular HAC partitioning gave the highest (α, β)-satisfaction for the random data. As we can see from the Figure 4.1, α-percentage of β-satisfied users on randomly generated data is significantly lower compared to the same results for Netflix movie ratings data. That shows us that, re-

Table 4.2: Best partitions found in experiments for the A-model

β	**max α%**	**HAC Method**	**Best Partition**
0.2	80.34	Single,Centroid	{9, 10, 11, 12, 13 14, 15, 16} {3} {1, 2, 4, 5, 6, 7, 8, 17, 18, 19, 20, 21, 22, 23}
0.4	59.76	Single	{9, 10, 11, 12, 13 14, 15, 16} {3} {1, 2, 4, 5, 6, 7, 8, 17, 18, 19, 20, 21, 22, 23}
0.6	27.08	Single,Centroid	{9, 10, 11, 12, 13 14, 15, 16} {3} {1, 2, 4, 5, 6, 7, 8, 17, 18, 19, 20, 21, 22, 23}
0.8	10.29	Single,Centroid	{9, 10, 11, 12, 13 14, 15, 16} {3} {1, 2, 4, 5, 6, 7, 8, 17, 18, 19, 20, 21, 22, 23}

garding the Netflix data, we are getting meaningful partitions that are able to satisfy much higher percentage of users compared to randomly generated, genre non-biased data.

The Figure 4.2 describes experimental results for the Jaccard error distance between partitions produced by the A-model algorithm and the human-based genre partitioning described in the Table 4. As we can see from the Figure 4.2, the minimal error distance from the human-based genre partitioning has the preference partitioning produced by applying the A-model algorithm on the HAC Weighted Average Link and HAC Centroid partitions. As we can see from the chart, the error is relatively constant for different values of β as long as $\beta < 0.8$. When β reaches the maximum value, the error distance for the partitioning produced by HAC Complete Link significantly

increases. That is due to the fact that the preference partitioning for the maximum β satisfaction value for this partitioning method partitions all movies into the same segment and gives us a trivial partition for $\beta = 0.8$. From this figure we can also notice that partitioning produced on randomly generated, non-biased data will have a significantly higher Jaccard error distance from the human-based genre partition. That tells us that in the Netflix movie ratings database there exists a bias towards partitioning preference to certain groups of movies according to their genres.

4.2 Results for the B-model

In running experiments for the B-model, we followed steps of the greedy algorithm described in the section 3.3. In this case, input value for the algorithm was α parameter. For different values of α and families of partitions produced by the six different hierarchical agglomerative methods, we computed subsets of users of size at least αn that maximize the average cost benefit per edge β. We did that over all families of partition, and chose a partition and subsets of users within αn boundaries that produces the maximum average β-satisfaction per edge,which is computed as $\beta = \frac{2MaxBenefit}{|I|(|I|-1)}$.

As we can see from the Figure 4.3 for smaller values of α, the β values are higher. This is because, the more rows we remove for a good partition, the more likely we are to increase total benefit in each step, yielding the higher

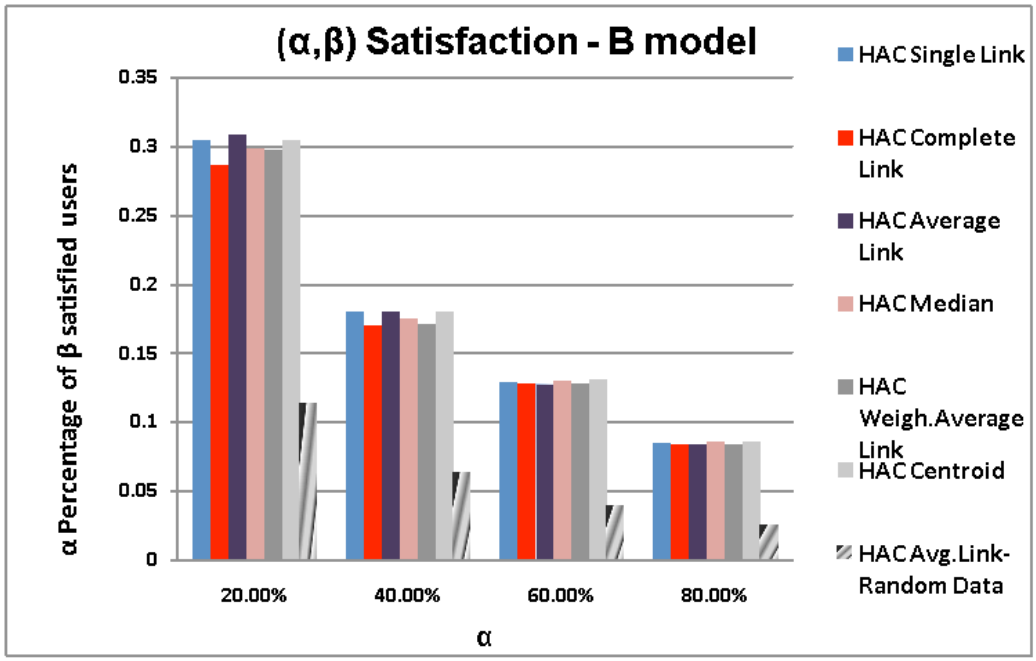

Figure 4.3: (α, β)-satisfaction results in B-model

average edge benefit β. The more users we remove for a good partition, the more likely we are to remove more noise from our data, therefore getting a very high level of the β satisfaction. For α values of $0.4n$, $0.6n$ and $0.8n$, we can see that all HAC methods yield very similar β values, and for lower values of α we can see that HAC Average Link method finds a partition that yields a little bit higher β-satisfaction,compared to other HAC methods. Similarly to experiments for the A-model,in the Figure 4.3 we also included experimental results that we obtained by applying our algorithm on HAC partitions applied on randomly generated data. We included only results produced by applying the B-model algorithm on partitions produced by applying Average Link HAC on the random data, since that particular HAC partitioning gave

Table 4.3: Best partitions found in experiments for the B-model

max β	α%	**HAC Method**	**Best Partition**
0.3085	20.00	Average	{1, 2, 4, 5, 6, 7, 8, 17, 18, 19, 20, 21, 22, 23} {3, 9, 10, 11, 12, 13 14, 15, 16}
0.1797	40.00	Single, Centroid	{1, 2, 4, 5, 6, 7, 8, 17, 18, 19, 20, 21, 22, 23} {3} {9, 10, 11, 12, 13 14, 15, 16}
0.1286	60.00	Single	{1}{19}{3}{4, 8}{11}{6}{21, 23}{10, 13} {17}{7}{2, 5}{14, 15}{12, 16}{9}{18}{20,22}
0.0863	80.00	Centroid	{1}{18, 19}{3}{4, 8}{11}{6}{21, 23}{10, 13} {17}{7}{2, 5}{14, 15}{12, 16}{9}{20,22}

the highest (α, β)-satisfaction for the random data. As we can see from the Figure 4.3, α-percentage of β-satisfied users on randomly generated data is significantly lower compared to the same results for Netflix movie ratings data. That shows us that, regarding the Netflix data, the B-model algorithm gives us meaningful partitions that are able to satisfy much higher percentage of users compared to randomly generated, genre non-biased data. If we analyze data in the Table 4.2 we can see that the best partitions for higher values of α (0.6 and 0.8) contain many small segments. These partitions yield relatively small β-satisfaction. For higher values of β-satisfaction the produced segments there are two partitions that maximize the average cost benefit per edge β, P_1 ={{ 1, 2, 4, 5, 6, 7, 8, 17, 18, 19, 20, 21, 22, 23},{3, 9, 10, 11, 12, 13 14, 15, 16}} and P_2 ={{1, 2, 4, 5, 6, 7, 8, 17, 18, 19, 20, 21, 22, 23}, {3},{9, 10, 11, 12, 13 14, 15, 16}}. These two partitions, similarly to the partitioning in the A-model, groups all black and white and all except one Woody Allen movies into one segment, and all James Bond movies into

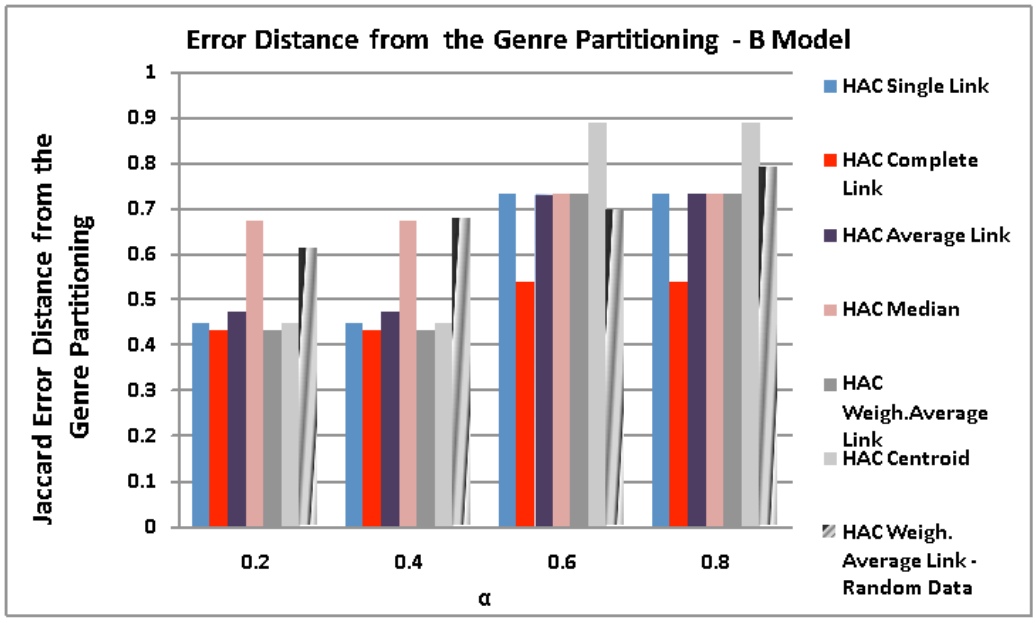

Figure 4.4: Measured error distance between produced preference partitions and the genre partitioning for the B-model

another segment. The item number 3 is relatively misplaced for the both partitions. It is a Woody Allen movie, but in the first partition it is classified into the James Bond movies segment, and in the second partition it is classified as a separate segment. When we analyzed ratings data available for this movie, we have noticed that there are many ratings missing for the movie (the ratings vector is relatively sparse). So we are assuming that the lack of ratings is the main reason of misplacing this item into the wrong segments. As we can see, for higher values of β-satisfaction, partition into segments tends to be closer to genre-based segmentation.

In the second sets of experiments for the B-model we measured tradeoffs

between the level of satisfaction β and the Jaccard error distance between produced preference partitioning and the human-based genre partitioning described in the Table 4. The smaller α values produce smaller cluster with higher β-satisfaction. Therefore, for the smaller values of α (0.2 and 0.4) the Jaccard error distance between the generated partitions and the genre-based partition is smaller for all HAC methods, compared to the Jaccard errors produced by partitions for higher α values. That is due to the previously mentioned observation that, for smaller values of α the B-method tends to produce partitions that are much closer to the human-based genre partition. Higher values of α produce partitions that are not very related to the genre partition, and therefore, they yield higher error distance from the genre partition. As we can see from the Figure 4.4, for the smaller values of α, partitions produced by Median HAC significantly under perform other HAC methods. For higher values of α (0.6 and 0.8) partitions produced by Complete Link HAC significantly outperform other HAC partitioning methods. In this figure we also included the measured Jaccard error distance between partitions produced on a randomly generated data and the genre partition. As we can see from the Figure 4.4, for the smaller values of α and higher values of β, Jaccard error distance between partitions on random data and the genre partition is much higher compared to the HAC methods applied on the Netflix data. For smaller values of α and higher levels of the β-satisfaction, all except one HAC methods produced significantly smaller Jaccard error distance from the genre partition, compared to partitions produced on the

random data. For higher values of α, the induced partitioning created much smaller β-satisfaction and under preformed in the terms of Jaccard error distance, compared to the random data. So we can see that, for the small values of α, in the clusters that have a higher average cost benefit per edge, β, the partitioning produced by the B-model algorithm is very close to the genre partitioning. However, for large values of α, where the β-satisfaction becomes very low, the produced partitioning is not much different from the partitioning on the random, genre un-biased data.

4.3 Results for Top-k Item Recommendation

For testing performance of the $top - k$ recommendation algorithm proposed in 3.5, we split our experimental data into the training and testing part. The training part contains around 90% of the all ratings from the chosen Netflix data subset described at the beginning of this chapter, and 10% of the ratings are removed and put into the testing data set.

We ran the set of experiments on the Netflix data, as described before, to test the quality of recommendations produced by our recommendation algorithm. As it was mentioned before the experimental dataset is split into two parts – the training set containing 90% of all ratings at the testing dataset containing the remaining 10%. The quality of the produced recommendations is compared to the $top - k$ recommendation algorithm produced by the

Nearest Biclusters Algorithm described in [31]. This algorithm is based on the biclustering process that is taking into account both users and items in a ratings data set. This algorithm uses binary data and keeps track only of the items that a given user likes (they have value 1) and all other items are considered as missing values (have value 0). The inputs for these algorithms are the minimum number of neighbors n and the minimum number of items k in a bicluster. It was reported in [31] that the Nearest Biclustering Algorithm exhibits the best behavior for $n = 4$ and $k = 10$, so we ran this algorithm with these parameters.

The quality of recommendation is measured by F_1 value, which is very common measure in information retrieval. The F_1 measure is given as

$$F_1 = \frac{2 \cdot recall \cdot precision}{recall + precision} \tag{4.2}$$

The recall and precision are given as

$$recall = \frac{hits}{items_{relevant}} \tag{4.3}$$

$$precision = \frac{hits}{items_{recommended}} \tag{4.4}$$

where $hits$ is the number of correct recommendations, $items_{relevant}$ is the number of all possible items that can be recommended for a give user and $items_{recommended}$ is number of items recommended to a user.

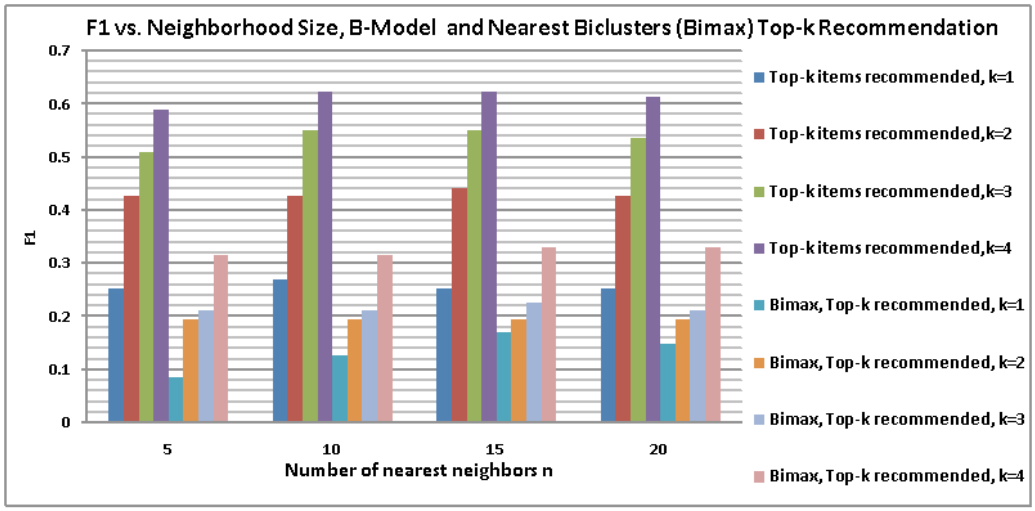

Figure 4.5: Quality of recommendations in the B-model based Top-k recommendation algorithm and the Nearest Biclustering based on Bimax

The experimental results from the Figure 4.5 show that, for all sizes of neighborhood, our $top-k$ recommendation approach outperforms the one given by the Nearest Biclustering Algorithm. We can also see that, for the neighborhood size of 10, our algorithm has the best $F1$ value. It means that, in our case, the ten closest neighbors might be the most relevant in making recommendation for missing ratings. Adding more than 10 neighbors into consideration when making recommendation can add more noise than relevant information.

4.4 Conclusion

In this thesis chapter we proposed a new collaborative filtering framework called the collaborative partitioning (CP) and the bi-criteria optimization

problem related to it. We introduced two models for evaluating users satisfaction by a given topics partition, and showed that the decision problems related to these two models are NP-complete. We approximated solutions for the decision problems in both models by generating partitions by using hierarchical agglomerative clustering algorithms, and proposed heuristical approaches for finding the best partition among all generated HAC partitions in both models. We evaluated the quality of our partitions on the subset of Netflix movie database ratings. We extended the proposed algorithms for CP to create a $top-k$ recommendation algorithm, and evaluate its performance against the Nearest Bicluster Algorithm, described in [31].

By analyzing results that we have obtained through testing our proposed approaches on the Netflix movie database ratings we came to some conclusions:
If β value in the A-model approximation algorithm is increased, a fraction α of β-satisfied users for the best partition will decrease. The same property holds when increasing α value in the B-model approximation algorithm. The increase of α-value will decrease the level of β-satisfaction for a computed α-fraction of users. This observation is in line with some properties of real data sets. In a real dataset we can rarely find a very large fraction of users who like and dislike items in a very same or a highly similar way. That does not mean that clusters of users with highly similar taste for items do not exist, but it is not realistic to expect for these clusters of users to be very

large.

We also came to a conclusion that approximation algorithms for the A and B models are able to detect existing preference towards certain sets of items, if that preference exists in a given dataset. That conclusion is made by comparing α and β coefficients computed by applying our approximation algorithms on a real dataset to those computed by applying the same algorithms on randomly generated data.

By analyzing Jaccard error distance of produced partitions a human-based genre partition we came to a conclusion that, in the B-model approximation algorithm, a higher α input will produce partitions that are less meaningful and similar to those produced on random data. However, for smaller α fractions of users ($\alpha \leq 0.4$) levels of β-satisfaction for users will be much higher, and those partitions will be much closer to a human generated genre-based partition.

By analyzing and comparing $F1$ values for $top - k$ recommendations produced by our proposed $top - k$ recommendation algorithm to those produced by Nearest Bicluster algorithm, we came to a conclusion that our recommendations produce recommendations of a higher quality compared to those generated by Nearest Bicluster algorithm. The reason for this can probably be found in a fact that our approach takes into account all existing rating (both positive and negative) when computing recommendations. Nearest Bicluster algorithm takes into account only positive data when computing recommendations.

Chapter 5

CF Framework for Finding Seminal Data

In this chapter we introduce a novel approach of collaborative filtering framework for finding *seminal* and *seminally affected work* for sets of items.

Seminal work, in general, includes previous work in some area that influenced the future work in the given area. In the terms of users, items and preference data, we use the concept of *seminal work* to mark items released in the past that are highly correlated to some future sets of items in the terms of users preferences. For example, for a given dataset of movie ratings, seminal work for a given set of movies includes movies released in the past, that are highly correlated either with the given set of query movies, or with movies that are highly correlated to that set.

Seminally affected work includes work in some area that is significantly influenced by some previously released work in the given area. *Seminally affected work* for a given set of items includes items released after the items in a given set that are highly correlated either with the given set of query items, or with items that are highly correlated to that set. For an example of a dataset of movie ratings, seminally affected work includes movies released after a given set of movies that are highly correlated with the given set of query movies, or with movies that are highly correlated to that set.

Examples of searches for seminal work are numerous, especially in citation analysis. For example, in [17] the authors are exploring citations available from CiteSeer to build graphs of citations. Two types of edges are used to represent relations between nodes (documents) in graphs : *Reference* and *IsReferencedBy*. The authors then use these graphs to find a partitions whose cut is minimal.

5.1 Mathematical Model for Finding Seminal and Seminally Affected Work

For a given users-items preference data, we build two correlation directed acyclic graphs (DAGs) $G_{chron} = (V, E_c)$ and $G_{rev} = (V, E_r)$. Nodes in V correspond to items form a given user-items preference data. The difference between G_{chron} and G_{rev} is in the direction of edges in the corresponding edges sets. In E_c, for two items (nodes) v_1 and v_2 from V, there exists a cor-

responding directed edge $\overrightarrow{e} = (v_1, v_2) \in E$ if the correlation between v_1 and v_2, $corr_{v_1,v_2}$ is above some relatively high threshold value $corr_{min}$, and the item represented by a node v_1 is released before the item represented by v_2. In E_r the edges are directed from items that are released later to items that are released earlier. So we can see that directions of edges are determined by a chronological order of the release times of items, from earliest to latest in G_{chron} and from latest to earliest in G_{rev}.

We used the Pearson-r correlation measure [25] to compute correlation values between pairs of movie items. This measure is very common in item-item collaborative filtering.
If U_c is a set of users who rated both items a and b, the Pearson correlation between these two items is expressed as:

$$corr_{a,b} = \frac{\sum_{i \in U_c}(v_{i,a} - \bar{r_a})(v_{i,b} - \bar{r_b})}{\sqrt{\sum_{i \in U_c}(v_{i,a} - \bar{r_a})^2 \sum_{i \in U_c}(v_{i,b} - \bar{r_b})^2}} \tag{5.1}$$

where $\bar{r_a}$ and $\bar{r_b}$ are mean vote values for items a and b.

Note that these approaches will not necessarily create rooted DAGs. However, it is possible to create a rooted DAG by adding an extra artificial root r.

5.2 Types of Queries for Seminal and Seminally Affected Items

There are three types of queries supported by the proposed approach on finding seminal and seminally affected items. All of them utilize the concept of finding seminal and seminally affected set of items to produce highly correlated recommendations for a given set of query items. One type of supported queries return for a given set Q of query items the seminal work items that are common for at least two items from the set. If the query set has only one item, it returns all seminal work items from the past that are highly related to the given item.

The second type of queries supported by our proposed approach are queries that return, for the given set Q of query items, all items released after the given query items Q that are commonly influenced by two or more items from Q. So in this case, we are looking for a set of future items which seminal work contains at least two items from the query set. If the query set has only one item, these types of queries return all items that are released after the given query item, for which the given query item presents the part of their seminal work. The third type of queries supported by our approach are queries that represent union of results for two previously discussed types of queries. So for a given set of query items Q this query returns all common seminal work items from the query set, as well as all items released after the items from the query set that were influenced by some subset of two or more

items from Q.

To be able to find seminal and seminally affected work for a given set of query items, we execute ancestor queries on DAGs described in 5.1. For a given set of query items Q we compute, for each item i, all ancestors of i in corrsponding correlation DAGs G_{chron}, G_{rev}, depending on what kind of queries we are executing. Once when all ancestors for all query items are computed, we find all common ancestors for items from the query set. Depending on a type of query, we search for common ancestors of query items in G_{chron} (for seminal work queries), G_{rev} (for seminally affected work queries) or in both (for union queries). The Algorithm 5.1 describes the procedure of looking for either seminal or seminally affected work.

Algorithm 5.1 Algorithm for computing seminal or seminally affected work for a set of items

1: INPUT: Set of Query Movies $Q = I_1, ..., I_k$ represented by their all-paths labels
2: OUTPUT: Set S=$I_{s_1}, I_{s_2}, ...I_{s_l}$ that represents the seminal work for the input set
3: Let *SemWorkSet=[]*
4: Let *AllAncestors= []*
5: **for all** query items q in Q **do**
6: A_{l_q}=all ancestor labels for q
7: *NewSemWork=AllAncestors*$\cap A_{l_q}$
8: *AllAncestors=AllAncestors*$\cup A_{l_q}$
9: *SemWorkSet=SemWorkSet*$\cup$*NewSemWork*
10: **end for**
11: Return *SemWorkSet*

To be able to execute ancestor queries in efficient ways, we propose com-

pact labelling schemes for DAGs to encode all nodes (items) in the each DAG such that each node's label contains all-paths information for the given node in the DAG. Besides keeping all path information for each node from a DAG, these schemes are space efficient and take relatively small space to store all-path information for each node. In that way, if a given seminal works query contains k items, we find all ancestors for each query item directly from a given all-path information label and compare it with the ancestors of other items.

5.3 Related Work

Several researchers have adopted correlation graph representations of the problem domain [9, 20]. In the collaborative filtering domain, [2] presents horting, a graph based technique where nodes represent users and directed edges between nodes correspond to the notion of predictability.

An NCA u of nodes v and w in a dag D is an ancestor of both v and w, where u does not have descendants that are ancestors for both v and w. Labelling schemes for NCA queries have been shown to belong to a larger, more general family, of graph labelling schemes that are called *the informative labelling schemes* (see [24]). This family includes all label-based graph representations that will allow retrieving certain specific global properties using only local information. Properties that have been studied include, subsumption check, descendants, ancestors or NCAs, and graph distances

[24].

[1] presents two tree labelling schemes for answering ancestor queries, with labels of size $\frac{5}{3}\log n + O(1)$ and $\frac{3}{2}\log n + O(\log\log n)$ bits respectively. These labelling schemes are used to label XML trees and provide efficient execution of the ancestor queries in XML web searching engines.

In Section 5.5, we describe the first phase of our encoding scheme by providing a simple and efficient algorithm for variable-length labelling of trees that ignores issues of delimiting edges. This provides a theoretical benchmark for our final labelling. In Section 5.6, we consider three potential delimiting schemes, and analyze bounds on the space required when applying each of these schemes. In Section 5.7, we show how these delimiting schemes can be used to extend the basic scheme on trees and thus compactly encode subgraphs of dags. In Section 5.8 we present some of the applications of the proposed labelling schemes on our movie recommendation system called MovieTrack.

5.4 Compact Labeling Schemes for Rooted DAGs - Problem Statement and Design Approaches

Formal Problem Statement: Given a dag $D = (V, E)$ (representing a hierarchical taxonomy) the problem of all local path encoding is defined as follows:

1) produce a labelling of the nodes of V, where each node v in V is represented by $L(v)$, a unique identifier in binary, 2) for each node v in V, produce an encoded binary string $E(v)$ so that all paths can be reconstructed from $E(v)$ only, where reconstruction means that for each path $r = v_1, v_2, \ldots, v_k = v$ from root r to node v we can present the associated sequences of node labels $L(v_1), L(v_2), \ldots, L(v_k)$ for each node on that path.

Our Design Methodology for All Local Path Encoding is divided into three stages. In the first stage we choose a spanning tree of the dag and produce a binary labelling of the edges of the spanning tree in such a way that the out edges associated with a node v are unique, and in addition we use the fewest bits possible to do this. Next, for each node v we concatenate the labels of the edges on the unique root to node v path; let $L'(v)$ denote this string. This labelling is not necessarily unique and thus can not be used as valid encoding. However, this is a quite compact labelling, requiring for each node v only $\lfloor logn \rfloor + \sigma_v$ bits, where n is the number of nodes in the dag and σ_v is the depth of v in the spanning tree (see Theorem 5.1). In the next stage, we produce a valid encoding of the nodes by solving the problem of delimiting each of the edges in $L'(v)$. There are several methods that can accomplish this, and we show that by considering the structure of the dag improved solutions are possible. The final stage of our methodology involves creating the final encoding $E(v)$ by creating a list of edges that do not lie on the spanning tree, yet are on some path from root r to v. Again various schemes are possible for encoding such an edge list, and we present several

alternatives.

We present several methods for the effective delimitation of edges. These various methods offer alternatives for fine tuning an encoding of all-paths information based upon specific parameter values of particular directed acyclic graphs.

We present a variable length encoding scheme (applicable for any tree) so that root to node v paths can be represented with $\sigma_v + \lfloor \log(n) \rfloor$ bits, where σ_v is the depth of node v in the tree. This result is before delimitation, which in general results in a constant factor overhead increase on the result above. We present a variety of delimitation schemes that can be tailored to specific dags and thus can minimize the constant factors involved.

We begin our encoding of a hierarchical taxonomy with a method using a prefix based approach in which all nodes are labelled with a string in such a way that for each node v there is a path to the root so that each node on this path is labelled with a prefix of the label used for v. We call this approach a greedy Dewey labelling due to its natural relation to Dewey decimal hierarchical classification commonly used for libraries.

The scheme is broken into two parts: the first is called the greedy Dewey labelling for trees (or, TGDL for short), and the second is called the extended greedy Dewey labelling for dags (or, EGDL for short).

5.5 Greedy Labelling for Trees (TGDL)

Our proposed DAG labelling [28] includes two schemes: the first is called the Greedy Dewey Labelling for Trees (or TGDL) and the second is called the Extended Greedy Dewey Labelling for DAGs (or EGDL).

The TGDL labelling scheme is a prefix-based labelling scheme. The first phase of the TGDL labeling algorithm includes finding and extracting a Breadth-First (BF) tree T from a rooted hierarchy H that represents some rooted taxonomy. After the BF tree is found, TGDL is performed as follows. The root r of the taxonomy (and the BFS tree T) is labelled with the label TGDL(r)=ϵ, where ϵ denotes an empty bit string. For each non-root node v in the tree T, TGDL(v)=TGDL(u)$\cdot$ edel(e_v)$\cdot$GDL(e_v), where u is the parent of v in T, e_v is the edge from u to v, and edel(e_v) is the edge delimiting label, which can be considered as encoding the length of GDL(e_v) (see Figure 5.1).

The GDL labels of the tree edges are obtained as follows. All children edges $e_{c_1}, , e_{c_j}$ of the parent node u are ordered in the non-decreasing order based on the size of the subtree rooted at the associated child. For each child edge e_{c_i} of the node v, the GDL label is assigned uniquely, where the i^{th} edge in the previously mentioned ordering is encoded with exactly $\lfloor \log_2(i+1) \rfloor$ bits. (see Figure 5.3).

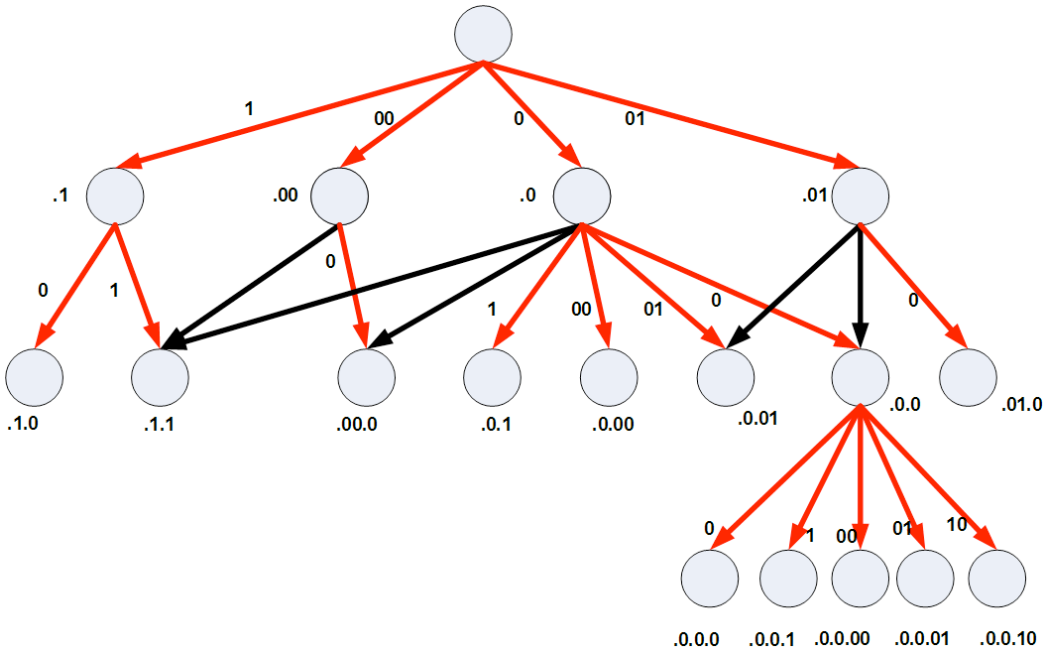

Figure 5.1: TGDL labelling of nodes of tree given by solid edges.

Analysis of the Length for the TGDL Labels The length of the TGDL labels depends, among other things, on the length and implementation of the edge delimiting labels. We performed analysis of the TGDL labels length in two steps. In the first step, we assumed that the edge delimiting labels are empty. This step is purely theoretical and we used it primarily to estimate the lower bound of our labelling approach. In the second step, we analyzed the length of the TGDL labels given different labelling schemes for the delimiting labels.

TGDL Labels Without Delimiters

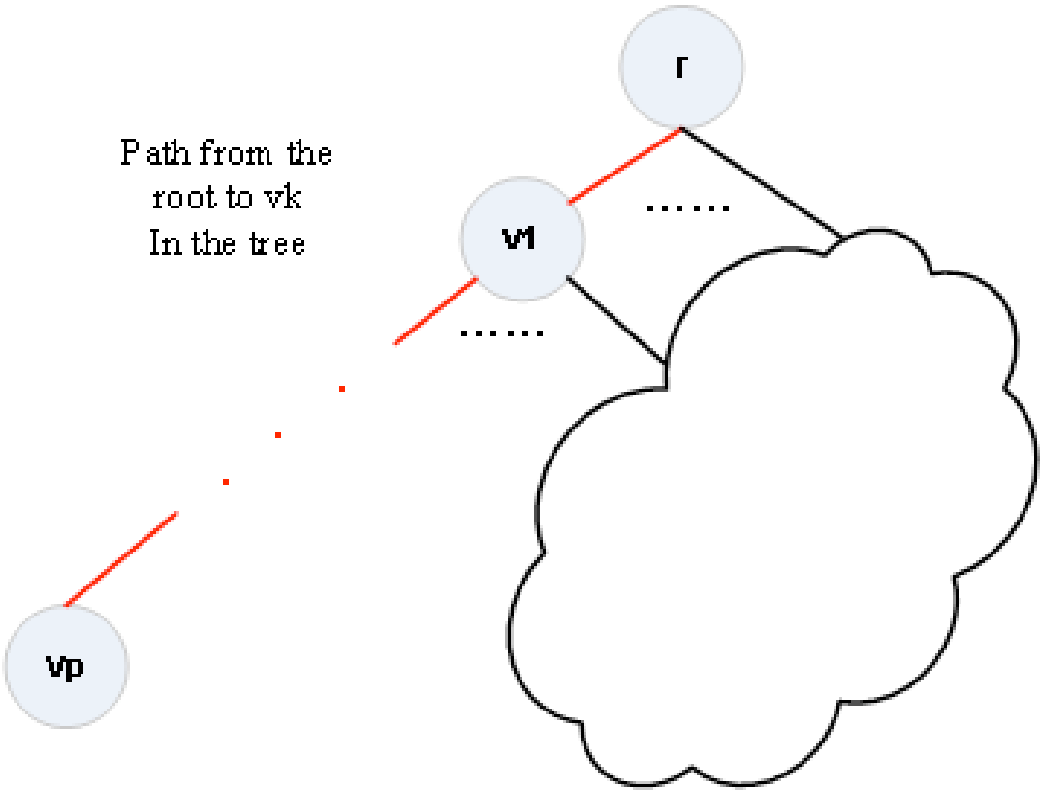

Figure 5.2: Path from a node to the root in a tree

Theorem 5.1. *Let T be a tree with n nodes. The TGDL algorithm labels each node v of the tree with at most $\sigma_v + \lfloor \log n \rfloor$ bits, where σ_v is the depth of v in T. This length excludes edge delimiters.*

Proof. Let n_v denote the size of the subtree of T rooted at node v. Consider any edge (u, v) in the tree T. We can bound number of bits used to encode this edge by the expression:

$$\lfloor \log(\frac{n_u}{n_v} + 1) \rfloor \leq \lfloor \log \frac{n_u}{n_v} + 1 \rfloor$$

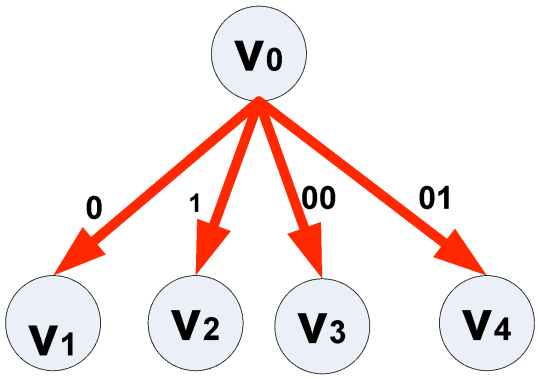

Figure 5.3: GDL labelling for tree edges

This is due to the fact that the GDL labelling uses exactly $\lfloor \log(i+1) \rfloor$ bits to encode the edge incident to the i^{th} largest subtree, and, for any i, the size of the i^{th} largest subtree is less than the size of tree divided by i.

Now consider any length-k tree path $v_0, v_1, , v_k$ from the root v_0 to the leaf node v_k. Obviously, this path is of length σ_{v_k}. As we assumed that the edge delimiting labels are empty, the total number of bits used to encode all edges on this path is bounded above by the sum:

$$\begin{aligned} \lfloor \log \frac{n_{v_0}}{n_{v_1}} \rfloor + 1 + \lfloor \log \frac{n_{v_1}}{n_{v_2}} \rfloor + 1 + ... + \lfloor \log \frac{n_{v_{k-1}}}{n_{v_k}} \rfloor + 1 \leq \\ \sigma_{v_k} + \lfloor \log \frac{n_{v_0}}{n_{v_1}} + \log \frac{n_{v_1}}{n_{v_2}} + ... + \log \frac{n_{v_{k-1}}}{n_{v_k}} \rfloor = \\ \sigma_{v_k} + \lfloor \log \frac{n_{v_0}}{n_{v_k}} \rfloor = \\ \sigma_{v_k} + \lfloor \log n \rfloor \end{aligned}$$

□

Algorithm 5.2 The TGDL algorithm

Input: Hierarchy H given as a dag D;
Output:TGDL labelling for T;

```
begin
Identify the breadth-first tree T in D;
Label root of T with an empty label, TGDL(r)=ϵ;
for all nodes v in T in breadth-first order do
  Perform GDL labelling for children edges of v in T;
  for all u_i that are children of v in T do
    Find edge delimiter edel(e_i) for e_i = (v, u_i);
    TGDL(u_i)= TGDL(v)·edel(e_i)·GDL(e_i);
  end for
end for
end.
```

5.6 Delimiting Schemes

We analyzed three different labelling schemes for edge delimiters, based on different methods for length encoding: Unary, fixed binary and variable binary scheme.

5.6.1 Unary Length Encoding

In the unary labelling scheme, for a given edge e, the edge delimiting label for e is a bit string of length $|e|$, $edel(e) = (0)_{|e|-1}1$, where $(0)_{|e|-1}$ denotes a $|e| - 1$ long zero bit string.

Corollary 5.1. *Let T be a tree with n nodes. If the unary length encoding scheme is used for encoding the edge delimiters, then the TGDL algorithm labels each node v of the tree with at most $2(\sigma_v + \lfloor \log(n) \rfloor)$ bits, where σ_v is*

the depth of v in T.

5.6.2 Fixed Binary Length Encoding

In the fixed binary labelling scheme, an edge delimiter for some edge e is the binary representation of the length for GDL(e). All edge delimiters are encoded with the fixed number of bits, that is approximately $\lceil \log \log(\Delta^*+1) \rceil$, where Δ^* is the maximum node out-degree that is found in a given tree.

Corollary 5.2. *Let T be a tree with n nodes. If the fixed binary length encoding scheme is used for encoding the edge delimiters, then the TGDL algorithm labels each node v of the tree with at most $\sigma_v + \lfloor \log n \rfloor + \sigma_v \lceil \log \lfloor \log(\Delta^* + 1) \rfloor \rceil$ bits, where Δ^* is the maximum node out-degree in T, and σ_v is the depth of v in T.*

5.6.3 Variable Binary Length Encoding

In this labelling scheme, for a given node v_p and a given path $rv_1...v_p$ from the root to v_p in a tree T, each edge delimiter in this path is encoded with approximately $\lceil \log \log(\Delta^*_{path_{v_p}} + 1) \rceil$ bits. $\Delta^*_{path_{v_p}}$ is the maximum node out-degree for the nodes on the given path. Additional bit string of length approximately of $\lceil \log \log \log(\Delta^* + 1) \rceil$ bits is preceding the edge encodings. This bit string is used to determine the maximum length of edge encodings at the given path.

Corollary 5.3. *Let T be a tree with n nodes. If the variable binary length*

encoding scheme is used for encoding the edge delimiters, then the TGDL algorithm labels each node v of the tree with at most $\sigma_v + \lfloor \log n \rfloor + \sigma_v \lceil \log \lfloor \log(\Delta^*_{pathv} + 1) \rfloor \rceil + \lceil \log \lceil \log \lfloor (\Delta^* + 1) \rfloor \rceil \rceil$ *bits, where* Δ^* *is the maximum node out-degree in* T*,* σ_v *is the depth of* v *in* T*, and* Δ^*_{pathv} *is the maximum out-degree for nodes on the path from the root to* v *in* T*.*

5.7 Extended Greedy for DAGs(EGDL)

Our goal for extending the 'greedy Dewey' labelling from the last section is to design a compact encoding for each node of the MovieTrack correlation DAG, so that all relevant path information can be effectively computed using only the information in the encoding of the node. Since we are, for this paper, considering the path structure of a dag D, we are interested in encoding for each node v of D the path-induced subgraph of v in D; call this $PS(v)$. This subgraph is defined as the unique minimal subdigraph of D that contains all the paths in the dag from the root to node v. Since our TGDL labelling implicitly encodes nodes as breadth-first paths from the root, we see that to encode $PS(v)$ we require only information about the edges that do not appear in the breadth-first tree. To generate the EGDL label for the node v, we simply concatenate together an edge list for all non-tree edges of the subgraph $PS(v)$.

Any edge of D can be represented by a pair of vertices. Therefore, in the EGDL labelling for D, each node v is represented as $EGDL(v) =$

$ndel(v) \cdot TGDL(v) \cdot edge_{EGDL}(e_1) \cdot ... \cdot edge_{EGDL}(e_g)$, where $edge_{EGDL}(e)$ is the EGDL encoding for the non-tree edges in $PS(v)$, $edge_{EGDL}(e)$ =ndel(v_1)·TGDL(v_1)·ndel(v_2)·TGDL(v_2), for $e = (v_1, v_2)$. $ndel(v)$ is a node delimiting label for v, and it can be implemented using the previously mentioned unary and fixed binary labelling schemes. In the fixed binary labelling scheme, the node delimiter for v is encoded with at most $\lceil \log(\sigma^*) \rceil$ bits, where σ^* is the depth of the BF tree of D.The unary labelling scheme labels each node delimiter for v with at most $\sigma_v + 1$ bits, where σ_v is the depth of v in the BF tree. Note that node delimiters are non-empty for each node, including the root.

Theorem 5.2. *Let m' be the number of non-tree edges in the subgraph $PS(v)$ of the Dag D, and let σ^* be the depth of a BF tree of D. The Extended Greedy Dewey Labelling (EGDL) for a dag D labels each node with at most $O(m'(\sigma^* + \log(n))$ bits.*

5.8 MovieTrack

The previously proposed approach for finding seminal and seminally affected work in items we applied to implement our movie recommendation system called MovieTrack [22].

The MovieTrack system is our web-based system based on movies from the Netflix movie ratings database that assists a user to find *seminal work* and *seminally affected work* for movies that he already liked. The intuition behind this approach is, if a user has a very high preference towards certain

sets of movies, that he will most likely be interested in some previous movies that are highly correlated with different subsets of the given movies.
It is 3-tiered system, meaning that it has a client application as the front tier (ASP.NET .aspx web page), set of XML .NET web services as the middle tier, and a MySQL database as the end tier (see 5.4).

The ASP.NET web page at the front end is used by a user to enter movies to perform one of three types of seminal work queries. Once when the items are entered, the client application calls one of the methods published by the given web service. The web service then queries the database, reads all paths information for each item in the seminal work query, evaluates all ancestor queries between each pair of items in the query set, queries the database to get titles for all computed ancestors and return results back to the user.

We store information from correlation DAGs in MySQL database. From all movies that are provided in the Netflix movie ratings database, we chose a subset of around 1000 highly correlated movies to build two correlation DAGs $G_{chron} = (V, E_h)$ and $G_{rev} = (V, E_r)$. The major drawback of the Netflix move ratings database is the sparsity of data. Unfortunately, we could not include those movie items that have really small number of ratings, as they are very poorly correlated to other items in the set.
For each node (movie) from the DAG we store the label produced by the Extended Greedy Dewey Labelling for DAGs (described in 5.7) using the

Variable Length Encoding Scheme. These labels enable us to perform ancestor queries on any subset of query items without querying the database multiple times.

5.9 Query Results and Conclusions

We performed some union queries using the MovieTrack system and here we present the results. Note that the DAGs we are querying are only 1000 nodes each, so they cannot include all possible seminal and seminally affected movies for a given set Q. When querying the system we use asp.net web page, that lists all 1000 movies from the DAG, with an options for each move to be checked by using checkboxes. For a snapshot of the MovieTrack front-end see 5.5. We pick sets of query items from different topics, as given by some lists from the Amazon Listmania [16]. Our sets of movies are chosen either by the topic (genre) or by a main actor in them.

As we can see from the Table 5.1, seminal and seminally affected works for a given set query are usually very similar, in some human, non-mathematical way of similarity, to the movies from the query set. There are three main topics to which query items belong to. The first group of query movies are movies that are considered to be classical American comedies with lots of American humor. As we can see from the Table 5.1, the seminal and seminally affected movies are also American comedies. The second group of query movies belong to the topic of "Sandra Bullock" movies. As we can see

from the Table 5.1, query results returned for this set of query items are also movies with Sandra Bullock in them. The third group of movies belong to the Listmania category of “dramedies”. All of these movies cover some very serious or dark subjects in a somehow comical way.We can see that the query with some dramedies produced more dramedies as well.

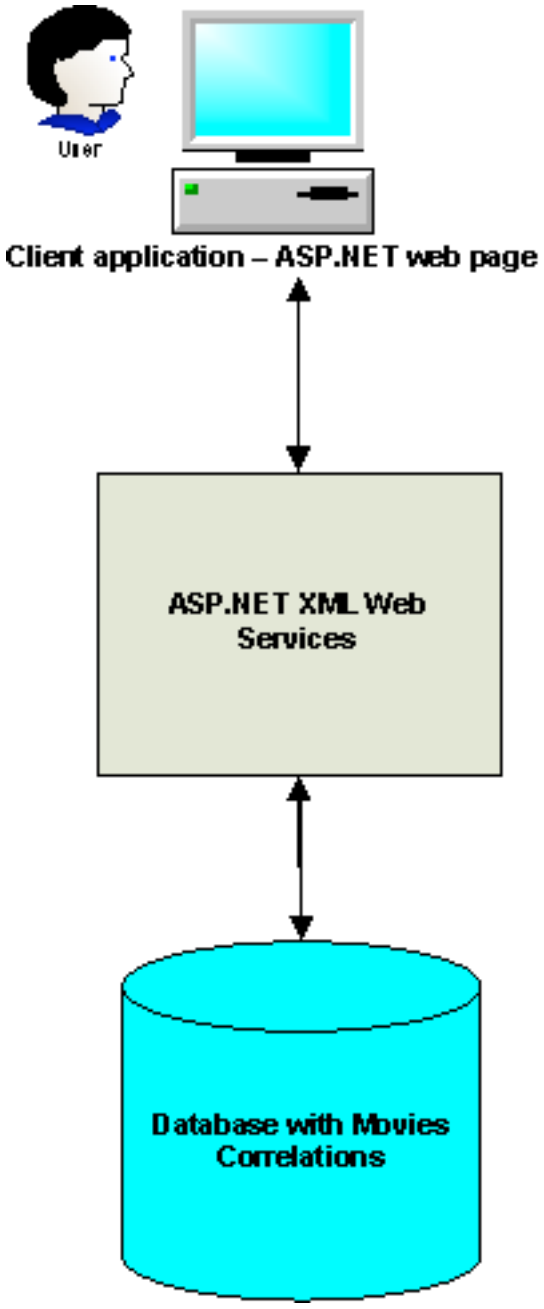

Figure 5.4: MovieTrack : 3-tier Architecture

Figure 5.5: MovieTrack : Front end GUI

Table 5.1: Results for union queries in the MovieTrack

Topic	**Query Items**	**Seminal and seminally affected work**
American Comedies	"National Lampoonś Van wilder" (2002) "American Pie" (1999) "American Pie" 2 (2002)	"Eurotrip" (2004) "The Girl Next Door" (2004) "American Wedding" (1999)
Sandra Bullock	"28 Days"(2000) "Murder By Numbers"(2002) "Two Weeks Notice" (2002)	"Practical Magic"(1998) "Hope floats"(1998)
Dramedies	"Adaptation" (2002), "The Royal Tenenbaums" "Eternal Sunshine of the Spotless Mind" (2004)	"Rushmore" (1998) "The Big Lebowski" (1998) "Being John Malkovich" (1999)
Romantic Comedies	"Noting Hill" (1999) "Runaway Bride"(1999) "You've Got Mail" (1998)	"Sweet Home Alabama" (2002) "Raising Helen" (2004) "Two Weeks Notice" (2002) "The Wedding Planner" (2001) "What Women Want" (2000) "How to Lose a Guy in 10 Days" (2003)

Chapter 6

Conclusions and Future Work

Conclusions and Contributions This thesis proposes novel frameworks and algorithms in the area of collaborative filtering. As we were able to experience while doing our research for this thesis, performing collaborative filtering tasks on large data sets is a very challenging task. Major two issues related to collaborative filtering algorithms are how to improve accuracy of and scalability. However, improving the performance of one of these two features usually negatively affects the other feature, so a real challenge in designing algorithms for collaborative filtering is how to design algorithms that exhibit a high performance for both of these features.

In the first part of this thesis we proposed a collaborative partitioning framework (CP) for partitioning items into segments that are able to satisfy a high percentage of users. We introduced the bicriteria, (α, β)- optimization

problem, where β is a coefficient expressing the level of a user satisfaction, and α is a coefficient that tells us what percentage of users are β-satisfied. We proposed two models for measuring quality of partitions in CP : the A-model and the B- model.
The bicriteria optimization problem introduced in this thesis is similar to the correlation clustering problem introduced in [4]. Both of the problems are NP-hard (we proved NP-hardness of our bicriteria optimization problem in section 3.4). However, finding a good approximation for our bicriteria optimization problem is more complicated, since edges in graphs are not labelled with simple $+, -$ labels, but with real numbers in the range $[-1, 1]$.
To measure quality of computed partitions we introduced two mathematical models - the A and B models.

The A-model measures the quality of partitions in CP by measuring number of threshold satisfied users. For a given partition $P = I_1, ...I_k$, a threshold satisfied user is a user that has the inner product with one (or more) of the segment vectors I_i in P greater than or equal to the threshold fraction of the square of its norm. So, in this case threshold satisfaction measures the level of agreement a given user with segments in P. The B-model measures the quality of partitions in CP by looking on the CP tasks as a graph clustering tasks. In this model CP is trying to maximize the sum of inter-cluster weights plus (negative) sum of intra-cluster weights. This model measures the quality of partitions by measuring how much benefit is

achieved by removing some users from a cluster in maximizing the previously mentioned sum.

As we proved in the Chapter 3, both optimization problems for the A-model and the B-model are NP-hard, so finding a scalable algorithmic solution for these problems is not an option. We present a heuristic polynomial time algorithms for approximating solutions in both models, where initial families of partitions are produced by using six different Hierarchical Agglomerative Clustering methods (HAC). As we showed in the experimental results, there are two types of conclusions that we have made analyzing experimentally produced partitions: 1.The partitions produced by the proposed heuristical algorithms are very close to a human-based genre partitioning, especially for highly satisfied users, and 2.The proposed heuristic approaches finds significant items partitioning in genre-biased Netflix data which are not found by applying the same heuristic approaches on randomly generated, genre non-biased data, and 3.CP algorithm for the B-model is a good starting point in creating a top-k recommendations for sets of users.

The novelty of our work is in the fact that we introduced new polynomial approximation algorithms for approximating CP problem, and we also define mathematical models to evaluate quality of achieved partitions. Additionally, we show that our CP approximation algorithm for the B-model can be used to design a $top-k$ recommendation algorithm that produces recommendation of higher quality, compared to other $top-k$ recommendation algorithms that work with similar data.

In the second part of this thesis we introduced a novel way of finding the seminal and seminally related work for a given sets of items, for a given users-items preference dataset. We introduced a method that translates item-item correlations into a correlation directed graphs (DAGs), where directions on edges are determined by a chronological order of release dates for movies (items). We proposed compact DAG labeling schemes that encode all path information for each node in a DAG, and show how these algorithms could be used for fast querying of the correlation DAG for seminal and seminally affected work for a given set of movies. As the experiments with the MovieTrack system show, search for seminal and seminally affected work for a given set of movies, tends to find highly related and relevant movies. The novelty of this project is that we proposed a graph model for representing preference datasets as directed graphs, and we also propose some novel approaches for efficient execution of ancestor queries on these DAGs, by designing compact labelling schemes for DAGs that encode all-path information for each node in relatively small node labels.

The contribution of this thesis to the areas of collaborative filtering is in finding novel collaborative filtering approaches that can be very useful in finding future item recommendations for users in a given users-items preference datasets. Algorithms that we propose in this thesis are scalable and suitable to be applied on very large datasets.

Future Work There are couple of research questions that should be considered in the future.
Regarding the collaborative partitioning, experiments should be performed to test connection between quality of partitions in the A-model with the quality of produced top-k recommendation lists. Future work also includes testing our proposed $top-k$ recommendation algorithm against some other $top-k$ recommendation algorithms, besides Nearest Cluster $top-k$ recommendation algorithm.

More experiments are needed in the future to experimentally evaluate how the proposed approximation algorithms perform when applied to real data sets of different nature. Also, more synthetic data should be generated, with one part of data generated in random, and one part containing non-random, genre-biased ratings, to see how the proposed approximation algorithms are going to perform in that case.

We noticed that a data sparsity can greatly affect the accuracy of partitioning in both A and B models. Our experimental results show that the items that are missing numerous reviews are usually misplaced in incorrect segments in CP, or are placed in a singleton segment. More research should be done in investigating if there is any way in removing this negative effect or automatically determining when a given item simply does not have enough data to be partitioned anywhere. Our CP proposed approaches are

working with "thumbs-up", "thumbs-down" and "no review" types of ratings only. The research question that should be investigated in the future is how to extend the proposed CP algorithms on working with a 5-scale ratings data.

In the research related to finding seminal and seminally affected work in movies, the future work involves incorporating a larger correlation DAG to work on (the current one contains around 1000 nodes), and testing how different correlation measures, besides the $Pearson - r$ correlation can affect the quality of results retrieve. More testing with human subjects is necessary to determine the human-evaluated accuracy of answers for seminal and seminally affected queries. So future work would include adding more functionality to MovieTrack system, that would give options to users to express their satisfaction and dissatisfaction with related movies found by MovieTrack. Also, it would be very interesting to research into possible connections between genre partitioning, that was covered in CP, and seminal work, because some of the queries show that they might be very connected (*i.e.* seminal work and seminally affected work queries on a same genre movie sets tends to return results in the same genre).

Bibliography

[1] S. Abiteboul, H. Kaplan, and T. Milo. Compact labeling schemes for ancestor queries. In *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 547–556, Washington DC, January 07-10 2001.

[2] C.C. Aggarwal, J.L. Wolf, K.L. Wu, and P.S. Yu. Horting hatches an egg: A new graph-theoretic approach to collaborative filtering. In *In Proceedings of the Fifth ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD99)*, pages 201–212, San Diego, CA, August 15-18 1999.

[3] Marko Balabanović. An adaptive web page recommendation service. In *AGENTS '97: Proceedings of the First International Conference on Autonomous Agents*, pages 378–385, February 5-8 1997.

[4] N. Bansal, A. Blum, and S. Chawla. Correlation clustering. *Machine Learning*, 55(1-3):89–113, 2004.

[5] Nicholas J. Belkin and Bruce W. Croft. Information filtering and information retrieval: two sides of the same coin. *Communication of ACM*, 35(12):29–38, 1992.

[6] James Bennet and Stan Lannin. The netflix prize. In *KDDCup '07*, August 2007.

[7] U. Brandes, M. Gaertler, and D. Wagner. Experiments on graph clustering algorithms. In *Proceedings of the 11th Annual European Symposium on Algorithms*, volume 2832, pages 568–579. Lecture Notes in Computer Science, September 15-20 2003.

[8] John S. Breese, David Heckerman, and Carl Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the Fourteenth Annual Conference on Uncertainty in Artificial Intelligence*, pages 43–52, Madison, Wisconsin, July 24 - 26 1998.

[9] P. Cohen and R. Kjeldsen. Information retrieval by constrained spreading activation on semantic networks. *Information Processing and Management*, 23(4):255–268, 1987.

[10] Brent J. Dahlen, Joseph Konstan, Jon Herlocker, Nathaniel Good, and John Riedl. Jump-starting movielens: User benefits of starting a collaborative filtering system with dead date. Technical Report 98-017, University of Minnesota, 1998.

[11] E.D. Demaine and N. Immorlica. Correlation clustering with partial information. In *Proceedings of the 6th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems*, August 24-26 2003.

[12] Google News Website. http://www.news.google.com, 2008.

[13] Elmar Haneke. 'learning' based filtering of text information using simple interest profiles. In *Proceedings of the First International Workshop on Cooperative Information Agents*, volume 1202, pages 171–182, Kiel, Germany, February 26-28 1997.

[14] J. Konstan, B. Miller, D. Maltz, J. Herlocker, L. Gordon, and J. Riedl. Grouplens: Applying collaborative filtering to usenet news. *Communications of ACM*, 40(3):77–87, 1997.

[15] Greg Linden, Brent Smith, and Jeremy York. Amazon.com recommendations - item to item collaborative filtering. *IEEE Internet Computing*, 7(1), feb 2003.

[16] Amazon Listmania. www.amazon.com/gp/richpub/listmania/toplists, 2008.

[17] G. Mermoud, M. Schaub, and G. Theoduloz. Partitioning citeseers citation graph. Technical report, Ecole Polytechnique Federale de Lausanne, Lausanne, Switzerland, 2005.

[18] Bradley Miller, John Riedl, and Joseph Konstan. Experiences with grouplens: Making usenet useful again. In *Proceedings of the Usenix Winter Technical Conference*, Anaheim, CA, January 6-10 1997.

[19] Bradley N. Miller, Istvan Albert, Shyong K. Lam, Joseph A. Konstan, and John Riedl. Movielens unplugged: Experiences with an occasionally connected recommender system. In *Proceedings of ACM 2003 Conference on Intelligent User Interfaces (IUI'03)*, Miami Beach, FL, Jenuary 12-15 2003. ACM.

[20] B. Mirza, B. Keller, and N. Ramakrishnan. Studying recommendation algorithms by graph analysis. *Journal of Intelligent Information Systems*, 20:131–160, 2003.

[21] Koji Miyahara and Michael J. Pazzani. Collaborative filtering with the simple bayesian classifier. In *Pacific Rim International Conference on Artificial Intelligence*, pages 679–689, Melbourne, Victoria, Australia, August 28 - September 01 2000.

[22] MovieTrack System. http://mybook.uc.edu/MovieTrackC/AllSem.aspx, august 2008.

[23] Netflix Prize Website. http://www.netflixprize.com, 2007.

[24] D. Peleg. Informative labeling schemes for graphs. Technical Report MCS00-05, Mathematics & Computer science, Weizmann institute Of Science, 2000.

[25] P. Resnick, N. Iacovou, M. Suchak, P. Bergstorm, and J. Riedl. Grouplens: An open architecture for collaborative filtering of netnews. In *Proceedings of ACM 1994 Conference on Computer Supported Cooperative Work*, pages 175–186, Chapel Hill, North Carolina, October 22-26 1994.

[26] G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Commununications of ACM*, 18(11):613–620, 1975.

[27] Upendra Shardanand and Patti Maes. Social information filtering: Algorithms for automating ẁord of mouth.́ In *Proceedings of ACM CHI'95 Conference on Human Factors in Computing Systems*, volume 1, pages 210–217, Denver, Colorado, May 07-11 1995.

[28] Svetlana Strunjas, Fred S. Annexstein, and Kenneth A. Berman. Compact encodings for all local path information in web taxonomies with application to wordnet. In *Proceedings of the Thirty Second International Conference on Current Trends in Theory and Practice of Computer Science*, pages 511–520, January 21-27 2006.

[29] StumbleUpon Social Media Website. http://www.stumbleupon.com, 2008.

[30] Chaitanya Swamy. Correlation clustering: Maximizing agreements via semidefinite programming. In *SODA '04: Proceedings of the Fifteenth*

Annual ACM-SIAM Symposium on Discrete Algorithms, pages 526–527, New Orleans, Louisiana, January 11–13 2004.

[31] P. Symeonidis, A. Nanopoulos, A. Papadopoulos, and Y. Manolopoulos. Nearest biclusters collaborative filtering. August 20–24 2006.

[32] P. Symeonidis, A. Nanopoulos, A. Papadopoulos, and Y. Manolopoulos. Nearest-biclusters collaborative filtering based on constant and coherent values. *Information Retrieval*, 11(1):51–75, 2008.

[33] Five Milion Users and Nearly Five Bilion Stumbles Later. http://www.techcrunch.com, April 23 2008.

[34] The end of theory: The data deluge makes the scientific method obsolete. http://www.edge.org/3rd_culture/anderson08/anderson08_index.html, june 2008.

[35] Tak W. Yan and Hector Garcia-Molina. The sift information dissemination system. *ACM Trans. Database Syst.*, 24(4):529–565, 1999.