GUARD: A Genetic Unified Approach for Recommendation

Adolfo Guimarães, Thales F. Costa, Anisio Lacerda, Gisele L. Pappa, Nivio Ziviani

Computer Science Department, Universidade Federal de Minas Gerais, Brazil
{adolfopg, thalesfc, anisio, glpappa, nivio}@dcc.ufmg.br

Abstract. Recommender systems suggest new items to users based on his/her interests. They appear in distinct contexts, such as movies, e-commerce, search engines, program guides for digital TV. In this work we propose a framework to generate ranking functions for items recommendation based on genetic programming, which we call GUARD (Genetic Unified Approach for Recommendation). The framework was developed under a collaborative filtering approach, but can be easily extended to work with content-based or hybrid recommender systems. The GP-based framework is flexible, works under a multi-objective approach and is able to combine different features and deal with data uncertainty and noisy. GUARD generates ranking functions considering four different measures: precision, recall, diversity and novelty. The framework was tested in the scenario of movies recommendation, using the Movielens 100k and Movielens 1M datasets. The results obtained were compared to those generated by PureSVD, the state-of-the-art algorithm for collaborative filtering. Considering the Movielens 100k, the results of precision and recall were superior to those of SVD. The framework can also generate more diverse and novel recommendations, with a small loss in precision. For Movielens 1M, the results are not as good as those obtained by PureSVD, but the functions slighly sacrifice accuracy over simplicity.

Categories and Subject Descriptors: H.3 [**Information Storage and Retrieval**]: Information Systems; I.7 [**Document and Text Processing**]: General

Keywords: collaborative filtering, genetic programming, recommender systems

1. INTRODUCTION

Searching and selecting Web content that users are interested in is an outgrowing challenge that asks for robust methods for content personalization and recommendation. In this scenario, recommender systems appear as an interesting solution for selecting relevant and useful information [Adomavicius and Tuzhilin 2005]. Recommender systems provide a suggestion of items according to user interests, where items can represent webpages, posts in social networks, movies, books and many other products.

The first step towards recommending an item is to define what the user interests are. User interests may be represented in many ways, depending on the recommendation approach followed [Ricci et al. 2011]. Recommender systems are usually based on three main approaches: (i) collaborative filtering, (ii) content-based or (iii) hybrid. In collaborative filtering, items are recommended to the user according to the preferences of similar users or considering similar items. The content-based approach recommends items with similar content to those already evaluated by the user. Finally, hybrid systems combine these two types of information [Adomavicius and Tuzhilin 2005].

Regardless of the approach followed, given a set of users and a set of items, recommender system can provide different outputs according to the objective of the system: it can be used to predict the rating a user is likely to give to an item, or it can generate a ranking of items, where the top-ranked items are assumed to be the most relevants to the user. This work focuses on collaborative filtering

We thank the Brazilian National Institute of Science and Technology for the Web (grant MCT-CNPq 573871/2008-6), Project MinGroup (grant CNPq-CT-Amazônia 575553/2008-1) and authors' grants and scholarships from CNPq and Fapemig.

approaches that generate lists of recommended items, which nowadays represent the state-of-the art in recommender systems [Cremonesi et al. 2010].

Collaborative filtering approaches are usually memory-based or model-based [Cacheda et al. 2011]. Memory-based approaches consider users or items similarity to suggest the lists of recommended items. One of the most used memory-based method is based on the cosine distance among an item/user and its/his/her k-nearest neighbors [Breese et al. 1998]. Model-based methods are mainly based on matrix-factorization techniques, such as Singular Value Decomposition (SVD) [Ricci et al. 2011]. Matrix-factorization techniques work by extracting latent factors that represent implicit relationships on data and are usually difficult to interpret [Koren 2008]. Hence, the recommendation models and the recommendations themselves cannot be explained to the user.

In this work we propose GUARD – Genetic Unified Approach for Recommendation, a new recommendation framework based on Genetic Programming (GP). The idea is to use a learning method to generate simpler memory-based ranking functions that represent interpretable models, which can provide an intuition behind a recommendation. The main motivation to create a GP-based framework to generate ranking functions for recommendation is its flexibility for combining features, besides its ability to deal with data uncertainty and noisy [Bäck et al. 2000].

GP is an evolutionary method popularized by Koza [1992] and based on the theories of evolution and survival of the fittest. It works with a population of individuals, where each individual corresponds to a candidate solution to the problem being tackled. The population evolves for a pre-defined number of generations, and evolution happens through crossover, mutation and reproduction operations. GP was already successful in other ranking learning tasks, such as obtaining rankings for search engines outputs [Fan et al. 2005] or associating ads to webpages [Lacerda et al. 2006]. However, to the best of our knowledge, so far GP and recommendation were not put together.

Although GUARD can be used to perform any of the three recommendation approaches aforementioned, this article focuses on collaborative filtering methods for movies recommendation. Hence, the ranking function evolved by GUARD is a combination of measures or (sub-)components of measures commonly used for collaborative filtering, such as the average rating of an item, item popularity, distance of user/item to the k nearest neighbors, among others. After being generated, the ranking functions are evaluated according to recommender systems measures, and then evolve with the application of genetic operators.

Although the main purpose of recommender systems is to maximize the accuracy of recommendation, i.e., to ensure good precision when predicting the user interest, other metrics can also be considered during this process [Herlocker et al. 2004]. This is important because the needs of the user may vary according to his/her profile. For instance, it is known that the age of the user in the system changes the type of answers he/she expects from the system. For new users, providing highly accurate recommendations is interesting for increasing his/her confidence in the system. However, for more experienced users, novelty and diversity are more relevant, even if they lower the accuracy of the recommendations [McNee et al. 2006]. This is because novelty and diversity surprise the users and keep them interested in the system. Maximizing these three metrics simultaneously is not a simple task, and has been studied in different contexts [Ribeiro et al. 2013]. GUARD can work with accuracy, novelty and diversity and their different combinations using a multi-objective approach.

GUARD was tested in two movie recommendation datasets, namely Movielens 100k and Movielens 1M, and compared to a memory-based method (based on cosine distance) and a model-based method (PureSVD). The results obtained in Movielens 100k are better than those of the state-of-the art algorithms, while for Movielens 1M the results are closer to those of PureSVD. However, an interesting fact showed by GUARD is that a simple and small subset of measures is extremely powerful to recommend items to new users.

This work is organized as follows. Section 2 introduces basic concepts related to recommender systems, which will be relevant when introducing GUARD. Section 3 reviews related work, while Section 4 describes GUARD. Section 5 presents experimental results using two well-known movie recommendation datasets. Finally, Section 6 presents conclusions and discusses future work directions.

2. BASIC CONCEPTS ON RECOMMENDER SYSTEMS

Recommender systems can be defined as a function $f : U \times I \rightarrow R$, where, $U = \{u_1, u_2, u_3, ..., u_m\}$ is the set of users, $I = \{i_1, i_2, i_3, ..., i_n\}$ is the set of available items, and R is a score that predicts the utility of an item to target users. Note that f is defined for *(user, item)* pairs that have not yet been seen by users. Conceptually, once the function f is estimated for all pairs of user and items, a recommender algorithm can select the item i_u^* with the highest score (or a set of N highest-scored items) for target user u and recommend item(s) to u, as follows:

$$\forall u \in Users : i_u^* = \arg\max_{i \in I} f(u, i)$$

Sarwar et al. [2001] divided collaborative filtering algorithms into two main categories – memory-based and model-based methods. Model-based algorithms recommend items using statistical or machine learning techniques to build a model of user preferences. On the other hand, memory-based algorithms generate a recommendation items list by using the whole user-item data to find (i) a set of users (i.e. *neighbors*) who rated different items similarly (user-based) or (ii) a set of similar items to the active item (item-based). In the next sections we detail these two approaches of collaborative filtering. The first will serve as bases for the set of terminals used by GUARD, while the second will serve as a baseline for the proposed framework.

2.1 Memory-based Algorithms

This section describes memory-based algorithms and presents two variants of this method, *user-based* and *item-based*. The criteria to choose an user-based or an item-based approach when considering memory-based algorithms depends on specific characteristics of the domain in which the recommender system will be applied. Specifically, when the number of items is greater than the number of users, item-based algorithms tend to obtain better accuracy than user-based methods. However, item-based methods use items similarity and hence have the drawback of suggesting less diversified lists of items. User-based algorithms are less sensitive to this problem [Burke 2002].

Regardless of the representation of users preferences that will be build when considering a memory-based approach, the recommendation task can be divided into three steps: (i) similarity computation, (ii) neighborhood selection, and (iii) rating prediction.

Defining how similar are users or items is a key aspect in memory-based recommender systems. Here we detail three methods to compute the similarity between pairs of users (cosine similarity, correlation-based similarity and constrained Pearson correlation similarity), followed by a metric that calculates item similarities.

Cosine-based Similarity. Consider that two users can be seen as two points in the n dimensional user-space. The similarity between them is computed as the cosine of the angle between these two vectors v. Specifically, the similarity between users a and u is given by:

$$s(a, u) = \sum_{j \in I} \frac{v_{aj}}{\sqrt{\sum_{k \in I_a} v_{ak}^2}} \frac{v_{uj}}{\sqrt{\sum_{k \in I_u} v_{uk}^2}} \quad (1)$$

where $a \in U$ is the target user and $u \in U$ is any other user in the dataset.

Correlation-based Similarity. Here, we use the Pearson correlation coefficient to measure the similarity between pairs of users:

$$s(a,u) = \frac{\sum_{i \in I_a \cap I_u} (v_{ai} - \overline{v_a})(v_{ui} - \overline{v_u})}{\sqrt{\sum_{i \in I_a \cap I_u} (v_{ai} - \overline{v_a})^2 \sum_{i \in I_a \cap I_u} (v_{ui} - \overline{v_u})^2}} \tag{2}$$

where $\overline{v_u}$ and $\overline{v_i}$ refer to the average rating value given by user u and the average rating value given to item i, respectively.

Constrained Pearson Similarity. Proposed by Shardanand and Maes [1995], this is a variant of the previous similarity metric that uses the central value of the rating scale rather than the average value. For instance, when considering a range rating scale as 1 to 5, the central value is 3. The similarity is given by:

$$s(a,u) = \frac{\sum_{i \in I_a \cap I_u} (v_{ai} - 3)(v_{ui} - 3)}{\sqrt{\sum_{i \in I_a \cap I_u} (v_{ai} - 3)^2 \sum_{i \in I_a \cap I_u} (v_{ui} - 3)^2}} \tag{3}$$

In contrast with the previous metrics, the similarity between a pair of items, based on the work of Sarwar et al. [2001], is given by:

$$s(i,j) = \frac{\sum_{u \in U} (v_{ui} - \overline{v_u})(v_{uj} - \overline{v_u})}{\sqrt{\sum_{u \in U} (v_{ui} - \overline{v_u})^2 \sum_{u \in U} (v_{uj} - \overline{v_u})^2}} \tag{4}$$

The second step in memory-based recommendation is to find the set of most similar users to the target user. The neighborhood of a user is composed by her/his N most similar users/items. Herlocker et al. [2002] empirically show that the best values for N range from 20 to 50 neighbors. Neighborhood setting is followed by the main step in memory-based methods, which is to compute the output rating of an item.

In user-based approaches, the predictions are directly related to the preferences of the N most similar users. One way to calculate the values of preferences is based on the normalization technique proposed by Herlocker et al. [2002], which determines the rating value of an item based on the average ratings of the user neighbors. This measure is adjusted according to the similarity between the target user a and his/her neighbors, as follows:

$$p_{ai} = \overline{v_a} + \frac{\sum_{u \in Neigh_a} [(v_{ui} - \overline{v_u}) s(a,u)]}{\sum_{u \in Neigh_a} s(a,u)} \tag{5}$$

For item-based approaches, the preference of target user a for an item i is related to the N most similar items to i and is given by Sarwar et al. [2001]:

$$p_{aj} = \frac{\sum_i (s(j,i) v_{ai})}{\sum_i |s(j,i)|} \tag{6}$$

Although memory-based methods are simple to implement, their performance is highly affected by sparse matrices of users and items. Furthermore, these methods are not easily adaptable to new users or items in the systems that have never evaluated or being evaluated, and hence do not have any neighbors in the dataset.

2.2 Model-based Algorithms

The design of statistical and machine learning models can help identifying and extracting complex relations from the training data, and then make intelligent predictions for test data. Different approaches, such as pLSA (*Probabilistic Latent Semantic Analysis*) [Hofmann 2004], neural networks [Salakhutdinov et al. 2007] and Singular Value Decomposition (SVD) [Burke 2002] have been proposed to solve the drawbacks of memory-based collaborative filtering algorithms.

In this work we focus on matrix factorization algorithms that represent the state-of-the-art in recommender systems, and have been shown to be effective to address the scalability and sparsity challenges of collaborative filtering tasks [Ricci et al. 2011]. Matrix factorization models map both users and items to a joint latent factor space of dimensionality k, such that user-item relations are modeled as inner products in this space.

The SVD method proposed by Koren [2008], for example, factorizes the user-item matrix into two factors: (i) each item i is associated with a vector $q_i \in \Re^f$ and (ii) each user u is associated with a vector $p_u \in \Re^f$. The inner product of this two vectors is used to predict the preference of u to item i as follows:

$$p_{ui} = b_{ui} + q_i^T p_u \tag{7}$$

where b_{ui} accounts for the user and item effects, and is given by $b_{ui} = \mu + b_u + b_i$, where μ is the overall average item rating. The parameters b_u and b_i stand for observed deviations of user u and item i, respectively, from the average. In order to compute b_u and b_i one can solve the least squares problem:

$$min_{b_*,q_*,p_*} = \sum_{(u,i)\in V} (p_{ui} - \mu - b_i - b_u - q_i^T p_u)^2 + \lambda(b_i^2 + b_u^2 + ||q_i||^2 + ||p_u||^2) \tag{8}$$

where λ is a regularization factor computed using cross-validation, μ is the overall average item rating and κ is the set of all user/item pairs (u, i) available in the training set.

The SVD method just described solves the rating prediction problem. In order to generate top-N items recommendation, Cremonesi et al. [2010] proposed PureSVD. When compared with SVD, PureSVD has more flexibility. For example, it considers all missing values in the user rating matrix as a given value, such as zero, despite being out of the 1-to-5 star rating range. That is possible because the value deviations given by this filling do not affect the relative order of the items in the ranking. The preference of user u for item i is predicted as:

$$p_{ui} = r_u \cdot Q \cdot q_i^T \tag{9}$$

where r_u is the line u of the user/item matrix and Q are the singular values extracted using SVD. PureSVD is the method used as baseline in the experiments performed with GUARD.

3. RELATED WORK

GP is well-known for its domain independence, tolerance to noise and implicit parallelism [Banzhaf et al. 1997]. By simply modifying the set of terminals and operations that define the individual and the fitness evaluation process, the same method can be applied to a solve a great variety of problems. For this reason, GP was already applied to many different areas [Poli et al. 2008], including information retrieval [Matos-Junior et al. 2012].

In the work of Fan et al. [2005], for example, GP was used to search for webpage rankings. The main objective of the problem was to combine different evidences traditionally used to rank documents according to user search preferences, including textual and structural features of the pages. Yeh et al. [2007] also used GP for Web search, but incorporated a greater number of evidences than those used by Fan et al. [2005]. Another example of work that generates ranking functions for information retrieval is that of Lacerda et al. [2006]. The authors used GP to learn how to associate ads to webpages. The work used a set of evidences related to ads and webpages in order to generate ad-webpages association functions. The best function found was 61% better than the baseline proposed.

Many works in content-based image retrieval also take advantage of the flexibility of GP. Torres et al. [2009] introduced a framework for generating functions that group different types of image descriptors in a robust way, finding results better than those obtained by state-of-the-art methods. Anand and Bharadwaj [2010], in turn, proposed the closest work to GUARD, where GP is used to

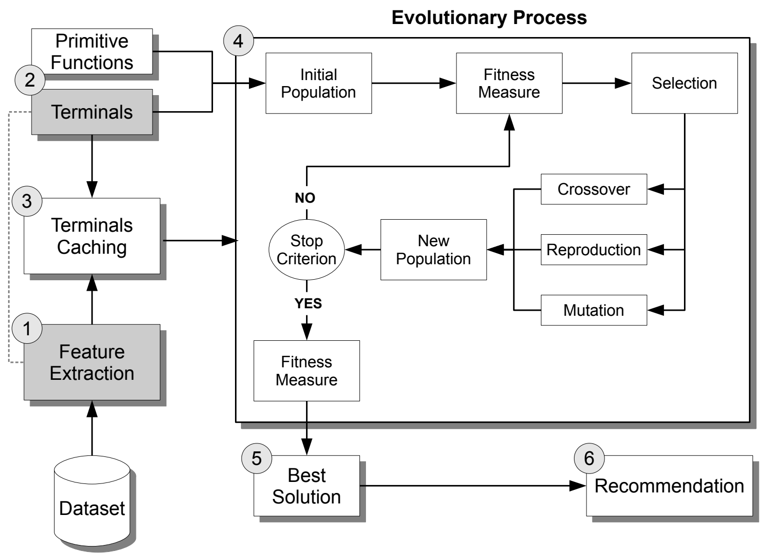

Fig. 1: GUARD: A GP framework for recommendation

find functions to transform raw ratings data into user preference data. The idea is to avoid problems with users that always evaluate an item with very high or very low rates. Once this bias is computed, it is used to calculate similarities between users in a collaborative filtering approach. Note that the work proposed here has a different goal, which is to find functions that return the final preference of a user regarding an item and, consequently, a ranking of the relevant items.

4. GUARD: A GENETIC UNIFIED APPROACH FOR RECOMMENDATION

This section introduces GUARD (Genetic Unified Approach for Recommendation), a framework based on GP for generating ranking functions for recommender systems. Here we focus on a collaborative filtering approach, although the framework is easily extendable to support other approaches. The framework gets as input a matrix M_{ui}, where u represents the number of users and i the number of items available. From this matrix, the algorithm generates a ranking function that order items according to their relevance to the user. The next sections discuss in detail the proposed framework, starting with a broader overview followed by the detail descriptions of phase 4, which encompasses the evolutionary process.

4.1 Overview

GUARD is a framework divided into six main phases that can be easily modified or extended. Figure 1 illustrates these six steps: (1) Feature extraction; (2) Definition of GP functions and terminals; (3) Terminals caching; (4) Evolutionary Process; (5) Choice of the best solution; (6) Recommendation. Steps that are gray-shaded in the figure require some type of interaction with the user, as described next.

The first step, feature extraction, selects from the dataset the relevant information for recommendation. The type of information extracted varies according to the approach being followed. In collaborative filtering, for example, the items evaluated by each user are considered, while content-based systems require description about items. In the second step, the functions and terminals that will

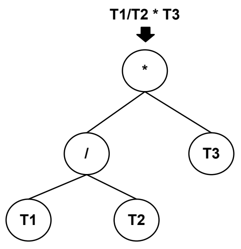

Fig. 2: Tree representation of an individual that corresponds to a ranking function

generate the first population of individuals (candidate ranking functions) are defined (see a detailed explanation in Section 4.2). The functions are usually standard mathematical operators, but the terminals vary according to the type of features being considered. After being defined, the terminals are calculated and cached in order to reduce computational time. The fourth step corresponds to the GP algorithm, which uses the initial population to start the evolutionary step. After evolution, a single solution has to be chosen to be used as a ranking function. This function is then used to recommend items to users.

In phase 4 we describe the evolutionary process. Recall that, in a GP algorithm, the initial population is created from a set of pre-defined terminals and operators (see Section 4.2). Each individual in the population represents a candidate ranking function. Following the creation of the population using the standard ramped-half and half procedure [Banzhaf et al. 1997], the individuals are evaluated, as described in Section 4.3. After evaluation, individuals are selected (see Section 4.4) to undergo crossover, mutation and reproduction operations according to user-defined probabilities (see Section 4.5). These new individuals form a new population, which will go though this entire process all over again, as illustrated in step 4 of Figure 1. At the end of the evolution, a single individual is chosen as the solution to the problem, as shown in Section 4.6.

4.2 Individual Representation

Although the individuals in GP can be represented by lists or graphs, the most common individual representation is the use of trees. In this case, the internal nodes of the tree represent one of the pre-defined operators, while the leaf nodes correspond to a terminal. Figure 2 represents the individual $T1/T2 \times T3$, where ($\times$) e (/) are operators and T1, T2 and T3 terminals.

In GUARD, the operators are represented by seven mathematical operations: sum (+), subtraction (-), multiplication ($\times$), division (/), power (*pow*), square root (*sqrt*) and logarithm (log). The terminals, in turn, depend on the recommendation approach being used. For collaborative filtering, the functions use information regarding the users evaluation towards items. For example, the dataset MovieLens has, for each user, the list of movies he/she evaluates and a rating varying from 1 to 5, which represents the preference of a user to a movie. This is the type of information the method explores to generate the ranking function.

Table I lists the terminals used by GUARD, which were mainly extracted from the similarity functions defined in Section 2.1. Consider $I = \{i_1, i_2, i_3, ..., i_n\}$ as the set of items, $U = \{u_1, u_2, u_3, ..., u_m\}$ the set of users, u_a is the user we are recommending to, v_{ai} the rating user a gave to item i, v_{ui} the rating user u gave to item i, $\overline{v_u}$ and $\overline{v_i}$ the average rating for user u and item i, respectively. Also consider $s1$ as the cosine similarity, $s2$ as the Pearson correlation coefficient and $s3$ as the Pearson correlation coefficient with constraints.

Terminals $T3$, $T4$ e $T5$ were originally defined by Koren [2008]. $T1$, $T2$ e $T6$ were extracted from the normalization metrics first proposed by Cacheda et al. [2011]. Terminals $T11$ and $T12$ were derived from the normalization metrics that use the users nearest neighbors defined by Koren [2008] and

Table I: Terminal list for collabortive filtering ranking functions

T1	Average rating value of user u
T2	Average rating value of item i
T3	b_u
T4	b_i
T5	b_{ui}
T6	Standard deviation of the user u ratings
T7	Number of users in the database
T8	Number of items in the database
T9	Number of items evaluated by user u
T10	Number of users that evaluated item i
T11_1	$\sum_{u \in Neigh(a)} (v_{ui} - \overline{v_u}) \times s1(a,u)$
T11_2	$\sum_{u \in Neigh(a)} (v_{ui} - \overline{v_u}) \times s2(a,u)$
T11_3	$\sum_{u \in Neigh(a)} (v_{ui} - \overline{v_u}) \times s3(a,u)$
T12_1	$\sum_{u \in Neigh(a)} s1(a,u)$
T12_2	$\sum_{u \in Neigh(a)} s2(a,u)$
T12_3	$\sum_{u \in Neigh(a)} s3(a,u)$
T13_1	$\sum_{j \in Neigh(i)} (v_{aj} - \overline{v_j}) \times s1(i,j)$
T14_1	$\sum_{j \in Neigh(i)} s1(i,j)$
TNUM	Random number varying from 1 to 5

Cacheda et al. [2011]. $T13$ and $T14$ use the nearest items and were also defined by Cacheda et al. [2011]. Finally, $T7$, $T8$, $T9$ and $T10$ are suggestions we make to improve the ranking functions.

4.3 Individuals Evaluation

After creating the individuals, the next step is to evaluate them. Here an individual is a function that, given a user a and and item i, returns the value of preference of user a to i. Having this preference value for all items in the dataset, we generate a ranking of items according to the preferences of user a. As each individual generates a ranking function, they need to be evaluated according to ranking evaluation metrics. Here four metrics are considered: precision, recall, novelty and diversity.

It is important to emphasize that, in recommendation tasks, the ranking evaluation can be a very complicated task because, in general, the number of items evaluated by a user is much smaller than the total number of items present in the dataset. Although traditional methods based on the presence of relevant items in the ranking are frequently used [Herlocker et al. 2004], here we follow the methodology proposed by Cremonesi et al. [2010]. In this work, the authors modify the way precision and recall are measured such that non-evaluated items can be considered. As the methodology for individuals evaluation is the same used in our final experiments, here we anticipate some concepts of the experimental methodology.

Evaluation Methodology. The methodology proposed by Cremonesi et al. [2010] divides the dataset into training and test sets. First, 1.4% of all ratings are selected to compose the *probe set* P and the rest of the data added to the training set Tr. Following, we select all examples in P with rating value greater than or equal to a given *score* to compose the test set Te. We use Tr to train the recommendation method and Te to test. For each example (u_j, i_j) in Te, we randomly select 1,000 items not yet evaluated by the target user u_j, creating a candidate list C with 1,001 items (i_j + 1,000 random items). The output of our algorithms, T, is a permutation of C sorted by the relevance of items to u_j and size N, i.e., we return the top$-N$ items of T.

Precision and Recall. The result of GUARD is a top$-N$ suggestion list of relevant items to users. Hence, we use two traditional metrics from information retrieval to evaluate these lists, i.e., precision and recall [Baeza-Yates and Ribeiro-Neto 2011]. However, we used the modified versions of these metrics as defined by Cremonesi et al. [2010]. Both metrics are based on the concept of *hit*. For each user u_j, the returned list of items contains the relevant item i_j and another 1,000 non-relevant items. We have a hit among the top$-N$ items if the rank position r_j of item i_j respects $r_j \leq N$, otherwise, i.e. if $r_j > N$, we have a *miss*. Based on this definition, the metrics of precision and recall are calculated as follows:

$$recall(N) = \frac{\#hits}{|Te|} \tag{10}$$

$$precision(N) = \frac{\#hits}{N \times |Te|} = \frac{recall(N)}{N} \tag{11}$$

Note that both metrics depend on the number of hits. Hence, instead of considering the precision and recall during the creation of the function ranks, we simply evaluate the number of hits.

Novelty and Diversity. Besides the accuracy of the returned lists, it is important for the user satisfaction that the recommended items are new and heterogeneous [Vargas and Castells 2011]. The recommended items list needs to be composed by items that the users would be unable to find by themselves and, at the same time, it is desirable that the suggested items are different from each other. In order to compute novelty and diversity, we follow the methodology presented by Vargas and Castells [2011], which looks at the position of the items in the recommended items list.

The novelty of an item takes into account its popularity in the whole dataset. The item popularity is given by the probability that a user would see an item and is computed as follows:

$$P(seen|i_k) = \frac{|u \in U | v(u,i) \neq \emptyset|}{|U|} \tag{12}$$

where U is the set of all users. Thus, the novelty of a top$-N$ list R to target user u is given by:

$$nov(R(N)) = EPC(N) = C \sum_{i_k \in R}^{i_N} disc(k)(1 - p(seen|i_k)) \tag{13}$$

where $disc(k)$ is a discount factor that considers the item position k in R and is computed as $disc(k) = 0.85^{k-1}$, and C is a normalizing constant given by $1/\sum_{i_k \in R}^{i_N} disc(k)$.

The diversity of a top$-N$ list R uses a model that takes into account the distance between the items in R and is given by:

$$div(R(N)) = EILD(N) = \sum_{i_k \in R i_l \in R l \neq k}^{i_N, l_N} C_k disc(k) disc(l|k) d(i_k, i_l) \tag{14}$$

where $disc(l|k) = disc(max(1, l-k))$ is also a discount factor that varies with the ranking position of items l and k, and $d(i_k, i_l)$ is a cosine similarity among a pair of items, given by:

$$d(i,j) = 1 - \frac{|U_i \cap U_j|}{\sqrt{|U_i|}\sqrt{|U_j|}} \tag{15}$$

where U_x is the set of users that liked item x.

Fitness Definition. Based on the methodology aforementioned, the fitness of an individual will be based on the three criteria which indirectly measure the four measures previously defined: number of hits (which influences precision and recall), novelty and diversity. In order to consider these objectives simultaneously, we introduce a multiobjective approach to GUARD. The multiobjective fitness can be implemented using three main approaches [Deb 2001]. The first one assigns weights to the objectives according to their relevance, and uses a single formula of weighted measures to attribute the fitness function. This approach main drawback is how to choose the weights for each objective.

The second and third approaches, namely Pareto-based and Lexicographic, do not join different values of objectives into a unique formula, and are the approaches used by GUARD. The Pareto-based approach compares individuals according to the concept of Pareto dominance. In a simplified way, a solution (individual) dominates another if it is better than the second in at least one objective and not worse than the second in all others. The lexicographical approach, in contrast, defines a priority order to the objectives. When comparing two individuals, the first objective in the priority list is taken into account. In the case of a tie (considering a standard deviation from the value of the objective), the second objective is analyzed, and this process goes on until the objective with least priority is evaluated [Poli et al. 2008].

GUARD uses the last two approaches when comparing the fitness of individuals during the selection process, which consider the three aforementioned objectives. Is is important to mention that there are many algorithms in the literature that were specifically proposed for multiobjective optimization. These algorithms were not considered here, and we simply changed the way the GP performs its selection process. Among these algorithms are SPEA2 and NSGA2 [Deb 2001], which will be incorporated to the framework in the near future.

4.4 Individual Selection

After individuals are evaluated, the selection phase starts by choosing the individuals that will undergo crossover and mutation operations. Here we describe two different selection schema adopted, generating different versions of GUARD. Regardless of the schema adopted, the selection process always starts with a tournament. Tournament selection is one of the most common selection process, where k individuals are randomly selected from the population and compete against each other [Bäck et al. 2000]. During this competition, comparisons among the individuals need to be made. The different schema just mentioned compare individuals in different ways.

In the Pareto-based approach, the individuals are compared using the three objectives simultaneously according to the Pareto dominance concept described above. The selection works as follows. Two individuals are randomly selected from the population, and compared to the k individuals selected for the tournament. The winner is the one which dominates the greater number of individuals in k [Deb 2001]. If both solutions dominate the same number of individuals, wins the one dominated by the smaller number of individuals.

For the lexicographical approach, we order the number of hits followed by diversity and novelty. Hence, the individuals are compared according to these measures in this order. The intuition is that rankings with high diversity and novelty are useless if they do not present a high precision.

4.5 Evolutionary Operations and Elitism

During evolution, three main operations are performed: crossover, mutation and reproduction. These operations are applied to the individuals selected with predefined user probabilities. In the case of crossover, two selected individuals exchange genetic material as follows. A node is randomly selected in each of the trees of the two individuals, and the subtrees below this node exchanged, generating two new individuals.

The mutation operations takes only one selected individual, which is randomly modified and generates a new one. GUARD implements two types of mutation: one-point mutation and shrink mutation. In the first case, a node in the tree is selected and replaced by a randomly generated one. If the chosen node is a terminal/function, it will be replaced by a new terminal/function. In the shrinking mutation, which aims to generate individuals smaller than the original ones, the selected node is always replaced by a terminal. Generating smaller individuals tends to improve the performance of the GP.

Finally, the reproduction simply passes an individual to the next generation with no modifications made. Note that the reproduction operation is different from the elitism, also used in the framework. The elitism preserves the best solutions of the current generation by adding them to the new generation with no added modifications. However, these solutions are not chosen by selection, but looking at all individuals of the population. When GUARD implements individual selection based on Pareto dominance, all the non-dominated individuals are inserted into the new generation, as long as their number is smaller than half the population size. In the case of the lexicographical approach, GUARD always preserves the individuals with the best values for hits, diversity and novelty.

4.6 Choosing the Best Individual

After the end of GUARD's execution, we have to select the best individual to be returned to the user. In an ideal world, the user of the system could meticulously analyze the solutions and choose the one that provides the best trade-off expected among the objectives. When this is not the case, automatically chosen this solution is not simple [Pedro and Takahashi 2011], and can be the subject of an entire paper. Hence, here we chose a very simple way to select the individuals, which is biased and does no guarantee the best choice was made. We return three individuals, each one with the top values for each of the objectives: number of hits, diversity and novelty.

4.7 A Note on GUARD's Time Complexity

The computational complexity of the algorithm depends highly on the cost of the fitness function. In GUARD, all terminals used during the fitness calculation process are pre-computed. In the worst case, the time complexity of this phase is $\mathrm{O}(i^2)$, where i represents the number of items in the dataset.

After this preprocess, each fitness evaluation costs $\mathrm{O}(i \times u)$, where u is the number of users. As GUARD evaluates n individuals for g generations, the total cost is roughly $O(i^2)+n\times g\times O(u\times i)$. More sophisticated methods, such as Markov Chain Models, can be used to estimate the time complexity of evolutionary algorithms taken into account the probabilities of applying genetic operators, but this is out of the scope of this paper.

5. EXPERIMENTAL RESULTS

This section describes the experimental evaluation of GUARD. We focus on movies recommendation, and tested the framework in two datasets: Movielens 100k and Movielens 1M. Movielens 100k has 943 users and 1,682 movies, and 100,000 user-item evaluations. Movielens 1M, in turn, presents one million evaluations performed by 6,040 users in 3,952 movies. In both datasets, the users explicitly demonstrate their preferences by assigning a number of starts to a movie that ranges from 1 to 5, where 5 is the best possible value.

5.1 Evaluation Methodology

The method is evaluated according to the methodology presented in Section 4.3. Recall that 1.4% of the rating in the dataset are selected and removed from the original dataset to form the probe base P. The remaining instances are added to the training set Tr. From P, all items with ratings considered relevant to the user (i.e., those with score 5, in the case of MovieLens) are added to the test set Te. The model is trained in Tr. During the test phase, for each item i in Te, 1,000 items not evaluated by the user are randomly selected from P, and considered not relevant to the user. These 1,001 items are then ranked, and the top$-N$ movies in the ranking evaluated using precision, recall, diversity and novelty. For all experiments we considered $N = 20$.

However, in order to first generate the function using GUARD and then test it into a new set of items, the training set Tr was subdivided into two other subsets: Tr' and a validation set Va, which is similar to Te and contains only relevant items. In this way, the GP is trained in Tr' (the training here corresponds to calculating the values of the terminals used by the ranking function), and along evolution the ranking functions have their fitness values calculated over Va. The best individuals, returned at the end of the evolution process, are then evaluated in Te. Three versions of GUARD are considered. GUARD-P and GUARD-P3 use a Pareto dominance criteria to calculate the fitness of the ranking functions being generated. However, GUARD-P takes into account only the number of hits, while GUARD-P3 also considers novelty and diversity. GUARD-Lex3, in turn, replaces the Pareto dominance approach by the lexicographical one, detailed in Section 4.3. The advantage of using the lexicographical approach here is that it will never neglect the values of precision over the values of diversity or novelty.

For the three versions of GUARD, we performed a series of tests for parameter setting. Some of these parameters vary from version to version and from dataset to dataset, but the following ones remained the same: 100 individuals are evaluated, and undergo crossover with a probability of 0.9 and mutation with a probability of 0.1. The number of generations the algorithms evolve for and the size of the tournament used to select the best individuals vary. For Movielens 100k, GUARD-P and GUARD-P3 run for 100 generations with tournament size equals 2. For GUARD-Lex3, 50 generations were executed using a tournament size equals to 20, in order to speed up convergence. For Movielens 1M, GUARD-P and GUARD-P3 also run for 100 generations, but the tournament size was set to 20. GUARD-Lex3 executed for 50 generations with tournament size equals to 3.

5.2 Computational Results

All results obtained by GUARD were compared with those obtained by two methods: one following a memory-based approach (see Section 2.1) and PureSVD. The memory-based approach uses a cosine similarity and 30 neighbors (see Eq. 5), and was chosen because it performs the same type of recommendation than the ranking functions generated by GUARD. PureSVD, on the other hand, represents the state-of-the-art algorithm for collaborative filtering. Here it was executed with 50 latent factors.

Table II: Comparing the average results of the best individuals obtained by three versions of GUARD for *Movielens 100k e 1M*. **H** indicates the returned individuals are the best in numbers of hits, **N** the best in novelty and **D** the best in diversity. Results in bold show cases where GUARD is significantly better than PureSVD. In the other cases, PureSVD is better than GUARD.

	Algorithm		**P@20**	**R@20**	**EPC@20**	**EILD@20**
Movielens 100k	Memory-based		0.0197	0.3939	0.7705	0.7944
	PureSVD		0.0266	0.5320	0.7999	0.8807
	GUARD-P	H	**0.0293 ± 0.0006**	**0.5859 ± 0.012**	0.7251 ± 0.008	0.7962 ± 0.019
	GUARD-P3	H	**0.0297 ± 0.0004**	**0.5933 ± 0.010**	0.7254 ± 0.009	0.8005 ± 0.018
	GUARD-Lex3	H	**0.0289 ± 0.001**	**0.5785 ± 0.021**	0.7363 ± 0.018	0.8076 ± 0.023
		N	0.0242 ± 0.004	0.4835 ± 0.097	0.8182 ± 0.071	0.8769 ± 0.062
		D	0.0242 ± 0.004	0.4848 ± 0.099	0.8176 ± 0.072	0.8810 ± 0.055
Movielens 1M	Memory-based		0.0186	0.3712	0.8232	0.8315
	PureSVD		0.0321	0.6427	0.8298	0.8881
	GUARD-P	H	0.0302 ± 0.0002	0.6043 ± 0.004	0.7717 ± 0.001	0.8412 ± 0.001
	GUARD-P3	H	0.0303 ± 0.0002	0.6056 ± 0.003	0.7762 ± 0.017	0.8422 ± 0.009
	GUARD-Lex3	P/R	0.0303 ± 0.00005	0.6069 ± 0.001	0.7686 ± 0.001	0.8374 ± 0.002
		N	0.0293 ± 0.0005	0.5855 ± 0.011	0.7834 ± 0.008	0.8601 ± 0.012
		D	0.0293 ± 0.0005	0.5855 ± 0.011	0.7834 ± 0.008	0.8601 ± 0.012

As GUARD runs a non-deterministic algorithm, each of the results reported is an average over 5 executions with different random seeds. Table II shows the average results obtained over five executions of GUARD and the two baselines considering Movielens 100k and Movielens 1M. For GUARD-P and GUARD-P3 the final individual selected is that based on the best number of hits in the validation set (H). For GUARD-Lex3, we show both the best individuals in number of hits (H) and those on novelty (N) and diversity (D).

The ranking functions generated by all versions of GUARD were always better than those obtained by the memory-based approach, which also presents results inferior to those obtained by PureSVD. These results were compared using Friedman's test, which will also be used throughout the comparisons in the remainder of this section. Given the poor results of the memory-based method, from now on, GUARD is always compared to PureSVD.

For Movielens 100k, note that the results of GUARD-P and GUARD-P3 are statistically better than those obtained by PureSVD, as showed by the results in bold in Table II. GUARD-P3, for instance, obtained a precision of 0.0297 against 0.0266 from PureSVD. Note that the results vary in the third decimal case. In order to give an idea of the results for the reader not used to the methodology proposed by Cremonesi et al. [2010], we also show the differences in terms of number of hits. As previously explained, a hit happens when a relevant item in *Te* appears in the top-20 ranking when ranked together with 1,000 items considered non-relevant to the user. In number of hits, the results show that GUARD-P3 puts 297 relevant items in the top-20 rank, 20 more than the number obtained by PureSVD, which represents a gain of 6.7% in number of hits. For GUARD-P3 we also returned the best individuals according the novelty and diversity. However, they are not reported here because they obtain values close to 1 for both metrics but with precision close to 0. A smarter way to select the individuals here need to be researched, but this is still an open problem in the area of decision making for multiobjective systems [Pedro and Takahashi 2011].

The problem above motivated the use of the lexicographical approach (GUARD-Lex3), which defines a priority order for the objective being optimized. Here, our priority is given by the number of hits followed by diversity and novelty. Note that, for these results, when selecting the best individual, there is no evidence of statistical difference when compared with PureSVD. The results for Movielens 1M, showed in the second half of Table II, show a different behavior of GUARD. Again, the results are better than those obtained by the memory-based approaches, but statistically worse than those obtained by PureSVD, where the difference is small but significant. Looking at the results, some hypothesis for the inferior results obtained by GUARD can be raised. The first one concerns the number of neighbors used when generating the ranking functions. In all cases, this number is set to 30. As Movielens 1M has more users, perhaps this number can have a big impact in the final results. The other hypothesis relates to the parameters of the GP, which were difficult to set in such a big dataset. A sampled dataset might help improving the parameter setting.

However, notice that the results presented so far are averages over 5 executions. In order to actually use the functions generated for recommendation, one of them needs to be selected. Table III shows the results obtained by the best of the five individuals in the validation set.

For Movielens 100k, the function produced by GUARD-P3 remains the best in terms of precision and recall. The best individual in terms of novelty and diversity is also better than PureSVD in these two criteria and very close in terms of accuracy. Considering the number of hits, PureSVD gets 32 more than GUARD-Lex3 (N or D) in a total of 297 recommendations performed, but GUARD-Lex3 improves the PureSVD novelty and diversity from 0.7999 and 0.8807 to 0.8421 and 0.9103, respectively. In this case, we can say that the function generated by GUARD-P3 is ideal for new system users, which prefer high precision over other characteristics in the recommendation results. In contrast, the function generated by GUARD-Lex3 is ideal for more experienced users, which prefer novelty and diversity over accuracy.

Table III: Comparing the best individual obtained by three versions of GUARD for *Movielens 100k e 1M*. **H** indicates the returned individuals are the best in number of hits, **N** the best in novelty and **D** the best in diversity. Results in bold show cases where GUARD is significantly better than PureSVD. In the other cases, PureSVD is better than GUARD.

	Algorithm		**P@20**	**R@20**	**EPC@20**	**EILD@20**
Movielens 100k	Memory-based		0.0197	0.3939	0.7705	0.7944
	PureSVD		0.0266	0.5320	0.7999	0.8807
	GUARD-P	H	0.0300	0.5993	0.7203	0.8017
	GUARD-P3	H	**0.0303**	**0.6061**	0.7235	0.8061
	GUARD-Lex3	H	0.0298	0.5960	0.7211	0.8043
		N	0.0249	0.4983	**0.8421**	**0.9103**
		D	0.0249	0.4983	**0.8421**	**0.9103**
Movielens 1M	Memory-based		0.0186	0.3712	0.8232	0.8315
	PureSVD		**0.0321**	**0.6427**	**0.8298**	**0.8881**
	GUARD-P	H	0.0306	0.6121	0.7739	0.8391
	GUARD-P3	H	0.0304	0.6086	0.7695	0.8378
	GUARD-Lex3	H	0.0304	0.6086	0.7688	0.8372
		N	0.0289	0.5777	0.7925	0.8708
		D	0.0289	0.5777	0.7925	0.8708

For Movielens 1M, the best results were obtained by GUARD-P, with a precision of 0.0306 against 0.0321 from PureSVD. In terms of number of hits, in a total of 3.168 recommendations, we got 90 hits less than PureSVD, which represents a difference of 3%. For the best individuals in terms of novelty and diversity generated by GUARD-Lex3, the difference in the number of hits obtained by PureSVD and GUARD-Lex3 is 206, which represents 6.5%.

Concerning the execution time of the algorithms, for Movielens 100k the processes of terminal caching, evolution and recommendation took an average time of 36 minutes and 05 seconds, being the recommendation itself of all items made in 5 seconds. For Movielens 1M, the total time increase to 19 hours and 57 minutes, where 9 hours were spent to cache terminals, 10 hours to evolve the ranking functions and 2 minutes to perform the recommendations. Recall that the first two phases are performed offline, and that third online. Note that the times reported used the methodology previously described, where the 297 users in the first dataset and 3,168 in the second are combined with other 1,000 items and ranked.

5.3 Generated Ranking Functions

In the previous section we analyzed the results obtained by the different versions of GUARD without looking at the ranking functions generated. This section focuses on the analysis of the terminals more frequent in the population of individuals and the best functions returned. For the sake of simplicity, we show only the functions generated by GUARD-P3. Although it generates more complex functions than GUARD-P (in terms of number of terminals), they are still simpler than the ones generated by GUARD-Lex3. The best ranking functions generated for Movielens 100k and Movielens 1M are given by Eq.(16) and Eq.(17), respectively.

$$r_{ai} = \log(T2_i \times T2_i + (T12\_1_a)^2 + \log((T12\_1_a)^2) + 2.08 \times T2_i) \tag{16}$$

$$r_{ai} = \log(2.35)/\sqrt[8]{(log(T10_i)/T2_i)}/T2_i \times (3.97 \times T12\_1_a)^2/T2_i \tag{17}$$

where $T10\_1_i$ is the number of users that evaluated i, $T12\_1_a$ is the sum of the cosine similarity among 30 nearest neighbors and $T2_i$ is the average number of ratings given to item i.

When analyzing the individuals returned, we notice that $T12\_1$ appears in all individuals. It is also present in most individuals of the population with a high value of precision. We also observe that when using the average rating of an item ($T2$) combined with T12_1 we obtain results close to or better than the ones obtained by PureSVD. This is certainly one of the main advantages of the framework: obtaining consistent results with simple, interpretable and explicit information.

In the same way that same terminals are important to generate the ranking functions, some others are quickly eliminated during the evolution of the algorithm, including $T12\_2$ and $T12\_3$, which represent the user similarity with his/her neighbors using the Pearson correlation and the constrained-Pearson correlation similarity, respectively, and $T1$, which is the average rating value user u attributes to the overall items.

6. CONCLUSIONS AND FUTURE WORK

In this work we present GUARD (Genetic Unified Approach for Recommendation), a GP based framework that generates items ranking functions for recommendation. The framework is flexible and can be instantiated for any type of recommendation, and evaluates candidate ranking functions according to three measures: number of hits (which is an indirect way of evaluating precision in recall), novelty and diversity. These three measures can be combined under two different approaches: a Pareto-based or a lexicographical approach.

The framework was tested under a collaborative filtering perspective in two datasets for movie recommendation: Movielens 100K and Movielens 1M, and compared to a memory-based method (based on the cosine distance) and a model-based (PureSVD). Three different versions of GUARD were tested, each of them considering different objectives in the fitness function. For Movielens 100k, the results of precision and recall obtained were statistically better than those obtained by PureSVD, which represents an increased of 7% in the number of items in the top-20 positions of the ranking. Concerning the ranking functions which were better at improving diversity and novelty, they decrease the values of accuracy. However, one could choose to use the first function to recommend to new users of the system and the second to more experienced users. Regarding *Movielens 1M*, the results of GUARD were not better than those obtained by PureSVD, and the average number of items in the top-20 positions of the ranking was 3% smaller. However, the great advantage of applying the ranking functions found is there simplicity associated with high values of precision.

As future works, we need to better understand why the results of *Movielens 1M* could not reach the values of PureSVD, start by testing the hypotheses raised in Section 5. Another promising direction is to propose more sophisticated ways of choosing the final solution returned to the user, and test some intelligent sampling techniques which can reduce the cost of training the model. In this same direction, we will investigate the possibility of using ranking aggregation of various evolved ranking functions to better improve the quality of the rankings generated. The framework will also be instantiated to perform content-based and hybrid recommendation. For that, a new set of terminals needs to be developed. Finally, we can easily modify GUARD's current version to perform rating prediction instead of generating the top$-N$ item ranking list, and to generate recommendations for other types of recommender systems that do not focus on movies, but music or products.

REFERENCES

Adomavicius, G. and Tuzhilin, A. Toward the Next Generation of Recommender Systems: a survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering* 17 (6): 734–749, 2005.

Anand, D. and Bharadwaj, K. K. Adaptive User Similarity Measures for Recommender Systems: a genetic programming approach. In *Proceedings of the IEEE International Conference on Computer Science and Information Technology*. Chengdu, China, pp. 121–125, 2010.

Bäck, T., Fogel, D., and Michalewicz, Z. *Evolutionary Computation 2: advanced algorithms and operators.* Institute of Physics Publishing, Bristol and Philadelphia, 2000.

Baeza-Yates, R. A. and Ribeiro-Neto, B. A. *Modern Information Retrieval - the concepts and technology behind search, second edition.* Pearson Education Ltd., Harlow, England, 2011.

Banzhaf, W., Nordin, P., Keller, R. E., and Francone, F. D. *Genetic Programming: an introduction (the Morgan Kaufmann series in artificial intelligence)*. Morgan Kaufmann Publishers, Heldelberg, 1997.

Breese, J. S., Heckerman, D., and Kadie, C. M. Empirical Analysis of Predictive Algorithms for Collaborative Filtering. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp. 43–52, 1998.

Burke, R. D. Hybrid Recommender Systems: survey and experiments. *User Modeling and User-Adapted Interaction* 12 (4): 331–370, 2002.

Cacheda, F., Carneiro, V., Fernández, D., and Formoso, V. Comparison of Collaborative Filtering Algorithms: limitations of current techniques and proposals for scalable, high-performance recommender systems. *ACM Transactions on the Web (TWeb)* vol. 5, pp. 2:1–2:33, 2011.

Cremonesi, P., Koren, Y., and Turrin, R. Performance of Recommender Algorithms on Top-N Recommendation Tasks. In *Proceedings of the ACM Conference on Recommender Systems*. New York, NY, USA, pp. 39–46, 2010.

Deb, K. *Multi-Objective Optimization using Evolutionary Algorithms*. Wiley-Interscience Series in Systems and Optimization. John Wiley & Sons, Chichester, Chichester, England, 2001.

Fan, W., Gordon, M. D., and Pathak, P. Genetic Programming-Based Discovery of Ranking Functions for Effective Web Search. *Journal of Management Information Systems* vol. 21, pp. 37–56, 2005.

Herlocker, J. L., Konstan, J. A., and Riedl, J. An Empirical Analysis of Design Choices in Neighborhood-Based Collaborative Filtering Algorithms. *IEEE Journal Information Retrieval* 5 (4): 287–310, 2002.

Herlocker, J. L., Konstan, J. A., Terveen, L. G., and Riedl, J. T. Evaluating Collaborative Filtering Recommender Systems. *ACM Transactions on Information Systems* 22 (1): 5–53, 2004.

Hofmann, T. Latent Semantic Models for Collaborative Filtering. *ACM Transactions on Information Systems* 22 (1): 89–115, 2004.

Koren, Y. Factorization Meets the Neighborhood: a multifaceted collaborative filtering model. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA, pp. 426–434, 2008.

Koza, J. R. *Genetic Programming - on the programming of computers by means of natural selection*. Complex adaptive systems. MIT Press, Cambridge, MA, USA, 1992.

Lacerda, A., Cristo, M., Gonçalves, M. A., Fan, W., Ziviani, N., and Ribeiro-Neto, B. Learning to Advertise. In *Proceedings of the International ACM SIGIR Conference on Research & Development of Information Retrieval*. New York, NY, USA, pp. 549–556, 2006.

Matos-Junior, O., Ziviani, N., Botelho, F. C., Cristo, M., Lacerda, A., and da Silva, A. S. Using Taxonomies for Product Recommendation. *Journal of Information and Data Management* 3 (2): 85–100, 2012.

McNee, S. M., Riedl, J., and Konstan, J. A. Being Accurate is not Enough: how accuracy metrics have hurt recommender systems. In *Extended Abstracts Proceedings of the Conference on Human Factors in Computing Systems*. New York, NY, USA, pp. 1097–1101, 2006.

Pedro, L. R. and Takahashi, R. H. C. Modeling Decision-Maker Preferences through Utility Function Level Sets. In *Proceedings of the International Conference On Evolutionary Multi-Criterion Optimization*. Ouro Preto, Brazil, pp. 550–563, 2011.

Poli, R., Langdon, W. B., and McPhee, N. F. *A Field Guide to Genetic Programming*. Lulu Enterprises, England, 2008.

Ribeiro, M. T., Lacerda, A., de Moura, E. S., Veloso, A., and Ziviani, N. Multi-Objective Pareto-Efficient Approaches for Recommender Systems. *ACM Transactions on Intelligent Systems and Technology* 9 (1): 1–20, 2013.

Ricci, F., Rokach, L., Shapira, B., and Kantor, P. B. *Recommender Systems Handbook*. Springer, New York, NY, USA, 2011.

Salakhutdinov, R., Mnih, A., and Hinton, G. E. Restricted Boltzmann Machines for Collaborative Filtering. In *Proceedings of the International Conference on Machine Learning*. New York, NY, USA, pp. 791–798, 2007.

Sarwar, B. M., Karypis, G., Konstan, J. A., and Riedl, J. Item-Based Collaborative Filtering Recommendation Algorithms. In *Proceedings of the International World Wide Web Conferences*. New York, NY, USA, pp. 285–295, 2001.

Shardanand, U. and Maes, P. Social Information Filtering: algorithms for automating word of mouth. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. New York, NY, USA, pp. 210–217, 1995.

Torres, R. d. S., Falcão, A. X., Gonçalves, M. A., Papa, J. P., Zhang, B., Fan, W., and Fox, E. A. A Genetic Programming Framework for Content-Based Image Retrieval. *Pattern Recognition* 42 (2): 283–292, 2009.

Vargas, S. and Castells, P. Rank and Relevance in Novelty and Diversity Metrics for Recommender Systems. In *Proceedings of the ACM Conference on Recommender Systems*. New York, NY, USA, pp. 109–116, 2011.

Yeh, J. Y., Lin, J. Y., Ke, H. R., and Yang, W. P. Learning to Rank for Information Retrieval Using Genetic Programming. In *Proceedings of the SIGIR Workshop: learning to rank for information retrieval*. Amsterdam, The Netherlands, 2007.