Retrieval of Collaborative Filtering Nearest Neighbors in a Content-Addressable Space

Shlomo Berkovsky, Yaniv Eytani, and Larry Manevitz

Computer Science Department, University of Haifa, 31905, Haifa, Israel
{slavax,ieytani,manevitz}@cs.haifa.ac.il

Abstract. Collaborative Filtering (*CF*) is considered one of the popular and most widely used recommendation techniques. It is aimed at generating personalized item recommendations for the users based on the assumption that similar users have similar preferences and like similar items. One of the major drawbacks of the CF is its limited scalability, as the CF computational effort increases linearly with the number of users and items. This work presents a novel variant of the CF, employed over a content-addressable space. This heuristically decreases the computational effort required by the CF by restricting the nearest neighbors search applied by the CF to a set potentially highly similar users. Experimental evaluation demonstrates that the proposed approach is capable of generating accurate recommendations, while significantly improving the performance in comparison with the traditional implementation of the CF.

1 Introduction

Recommender Systems [16] are one of the commonly used approaches to address the Information Overloading problem. They systems assist a user in selecting a suitable item among a set of potentially selectable items by predicting the user's opinion on the items [19]. Currently, Recommender Systems are used in a variety of application domains, such as movies [7], jokes [6], and music [1], and they exploit several recommendation techniques, such as Collaborative [9] and Content-Based Filtering [12], Case-Based Reasoning [17] and many hybrid techniques [4].

Collaborative Filtering (CF) is probably the most familiar and one of the most widely-used techniques to generate predictions in Recommender Systems. It relies on the assumption that people who agreed in the past will also agree in the future [21]. The input for the CF algorithm is a matrix of users' ratings on a set of items, where each row represents the ratings provided by a single user and each column represents the ratings provided by different users on a single item. CF aggregates the ratings to recognize similarities between users and generates the prediction for an item by weighting the ratings of similar users on this item.

The CF algorithm is typically partitioned to three generic stages: (1) Similarity Computation: weighting all the users with respect to their similarity with the *active user* (i.e., the user, whose ratings are being predicted), (2) Neighborhood Formation: selecting the most similar users for the prediction generation, and (3) Prediction Generation: computing the prediction by weighting the ratings of the selected users.

One of the major drawbacks of the CF is its limited scalability. The stages of Similarity Computation and Neighborhood Formation require comparing the active users with all the other users over all the available ratings. Hence, the computational effort required by the CF grows linearly with the number of users and the number of items in the ratings matrix. Thus, for a matrix containing ratings of *M* users on *N* items, the required computational effort is *O(MN)*. This poses a problem in systems, where the predictions are generated using millions of ratings on thousands of items, e.g., in Web-based Recommender Systems. Previous studies, (e.g., [3], [6], [5], and others) tackle the issue of reducing the computational effort required by the CF either by pre-processing of the ratings matrix or by distributing the computational stages. Nonetheless it remains one of the most important issues in the CF research community.

In this work we develop a fast heuristic variant of the CF algorithm that decreases the computational effort required by the Similarity Computation and the Neighborhood Formation stages. The basic assumption of the proposed heuristic algorithm is that losing general completeness of the exhaustive search (1) has a minor negative effect on the accuracy of the predictions, but (2) significantly decreases the required computational effort. Thus it provides a scalable approach, applicable to real-life scenarios with a high number of users and items, such as in Web-based systems.

The proposed heuristic approach is based on a notion of content-addressable data management [15] that provides an adaptive topology for mapping of users' profiles to a multi-dimensional space. This mapping implicitly clusters similar users and limits the Similarity Computation and the Neighborhood Formation stages to a heuristic search among the users that are potentially highly similar to the active user.

Experimental evaluation of the proposed approach demonstrates both high efficiency and good accuracy of the proposed algorithm in comparison with the traditional (exhaustive) *K-Nearest Neighbors* (KNN) search of the Neighborhood Formation stage. The evaluation also demonstrates that the algorithm is highly scalable with the number of nearest neighbors to be retrieved.

The rest of the paper is organized as follows. Section *2* describes the CF personalization technique and surveys the studies on the required computational effort reduction. Section *3* describes the CAN, a Peer-to-Peer content-addressable platform for decentralized data management. Section *4* describes the decentralized storage of users' profiles over the CAN platform and elaborates on the proposed heuristic variant of the CF over CAN. Section *5* presents and analyzes the experimental results. Finally, section *6* lists our conclusions and presents some open questions for future research.

2 Collaborative Filtering

Collaborative Filtering (*CF*) is probably one of the most familiar and widely-used recommendation techniques. An input for the CF is the so-called *ratings matrix*, where each user is represented by a set of explicit ratings given on various items, and each item is represented by a set of ratings given by the users. CF requires a similarity metric between users to be explicitly defined. The state-of-the-art CF systems exploit three similarity metrics: Cosine Similarity [7], Mean Squared Difference (MSD) [13], and Pearson correlation [19]. This work focuses on the MSD, computing the degree of similarity between users *x* and *y* by:

$$sim_{x,y} = \frac{\sum_{i=1}^{|x \cap y|} (R_{x,i} - R_{y,i})^2}{|x \cap y|}$$

where $|x \cap y|$ denotes the number of items rated by both users (typically, above some minimal threshold), and $R_{x,i}$ denotes the rating of user x on item i. In some sense, $sim_{x,y}$ can be considered as the *dissimilarity* of the users, as the lower the result of the MSD computation, the greater is the real similarity between the users. Prediction $P_{a,j}$ for the rating of the user a on item j is computed as a weighted average of the ratings of his/her K most similar users, i.e., *K nearest neighbors*, by:

$$P_{a,j} = R_a' + \frac{\sum_{k=1}^{K} (R_{k,j} - R_k') \cdot sim_{a,k}}{\sum_{k=1}^{K} |sim_{a,k}|}$$

where $R_{x,y}$ denotes the rating of user x on item y, R_z' denotes the average rating of user z, and $sim_{v,u}$ denotes the level of similarity between users v and u.

The Similarity Computation stage of the CF requires comparing the active user with every other user in the system. For a ratings matrix storing the ratings of M users on N items, the computational complexity of the Similarity Computation stage is $O(MN)$. This indicates poor scalability of the Similarity Computation, as the complexity grows linearly with both the number of users and the number of items in the matrix. Many prior works have dealt with decreasing the computational effort required by the CF. In general, it is achieved either by preprocessing the ratings matrix, or by distributing the computationally intensive stages of the CF among multiple machines.

Various pre-processing techniques for decreasing the computational effort required by the CF (e.g., correlation coefficients, vector-based similarity, and statistical Bayesian methods) are discussed and analyzed in [3]. Another technique, exploiting pre-clustering of the ratings matrix, is discussed in [6]. There, principal component analysis is used to identify two *discriminative* dimensions of the ratings matrix and all the vectors are projected onto the resulting plane. This inherently partitions the users to clusters or neighborhoods, which are further used to generate the predictions. In [5], the authors use a tree-like data structure and apply a *divide-and-conquer* approach using an iterative K-means clustering to group the users. This leads to smaller and more homogeneous clustering of users for the following Predictions Generation stage.

An alternative approach is to distribute the CF computational effort among the users, such that every user independently computes its similarity with the active user. This approach was initially proposed in [22] and elaborated on in [20]. The latter also developed a detailed taxonomy of the CF distribution approaches and presented implementation frameworks for different application domains. The PocketLens project [11] compared five decentralized distributed architectures for the CF. They showed that the performance of the decentralized mechanism is similar to the performance of the centralized CF while providing increased robustness and security. Further improvements to the decentralized CF were discussed in [8], which proposes the exploitation of Peer-to-Peer platform for a decentralized management of users' profiles. However, this approach approximates the set of the most similar users identified by the Neighborhood Formation stage of the CF, and as a result, the accuracy of the generated predictions is reduced.

This paper is loosely based on the ideas of CAN [15], a content-addressable Peer-to-Peer platform. We implement a fast heuristic variant of the CF, using a CAN-like multi-dimensional space for maintaining a connected structure of users. This allows to significantly decrease the computational effort required by the Similarity Computation and Neighborhood Formation stages by limiting the search process to a search among potentially similar users located in close vicinity to the active user.

3 Content-Addressable Data Management

This section presents the general architecture of CAN [15], a scalable decentralized data management platform. In CAN, the users are represented in a one-to-one manner by the nodes of a virtual N-dimensional coordinate space such that the location of the user's node is denoted by a vector $(v_1,v_2,\ldots,v_N)$, where v_i represents the numeric coordinate of the node within a dimension number i. In addition to the node, each user continuously manages an N-dimensional subspace, called a *zone*. For example, consider a 2-dimensional space partitioned to 3 zones, managed by users A, B, and C (figure 1-left). Note that the figure shows only the zones managed by the users, whereas the nodes themselves are not shown.

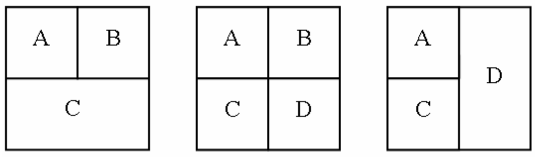

Fig. 1. Example of a *2*-Dimensional CAN space

In CAN space, two nodes (and zones) are called *neighbors* if their coordinate spans overlap along N-1 dimensions and adjoin along one dimension. For example, consider the neighbor zones A and C in figure 1-left, whose coordinates partially overlap across the horizontal dimension and adjoin along the vertical. To maintain connectivity in CAN space, each node stores a list of pointers to a set of other nodes, managing the neighbor zones. For example, node A stores pointers to the nodes managing zones B and C (as, respectively, horizontal and vertical neighbors) in its list of pointers.

Routing of messages in CAN space is based on the Plaxton routing algorithm [14]. This routing iteratively forwards the messages to the nodes that are closer to the target node than the current node using a greedy forwarding. The metric for evaluating the distance between two nodes in the address space is the L_1 metric, i.e., the Manhattan Distance. This metric was chosen due to the fact that CAN space inherently supports it, as every node stores a list of pointers to the nodes, managing the neighbor zones. For example, the distance between the nodes $(1,2,3)$ and $(6,5,4)$ in 3-dimensional CAN space is $(6-1)+(5-2)+(4-3)=9$. Thus, in N-dimensional CAN space a message is routed between an arbitrary pair of nodes in $O(N)$ routing steps.

In addition, CAN provides a connectivity maintenance algorithm, stable to sporadic joins and departures of new users. When a new user is inserted, it is assigned its own node and the respective zone. This is done by splitting a zone (determined by the content provided by the recently inserted user) of one of the existing neighbors according to the following steps: (1) the new user identifies an existing network node, (2) the new user is routed to the target zone that will be split, and (3) the target zone is split and the neighbors of the new zone are updated to maintain connectivity and facilitate future routings. As a result, only a subset of immediate neighbor zones of the zone that was split is actually affected by the insertion of a new node.

The issue of splitting the target zone (i.e., how to split the existing zone, where the contents of the recently inserted node are mapped?) is one of the important issues affecting the performance of CAN. A number of splitting policies are proposed, analyzed and compared in [15]. The simplest policy for the zones splitting is so-called *ordered* splitting. According to this policy, the number of dimension, across which a zone is split, iteratively increases from *1* to *N*. For example, consider user *D* joining CAN *2*-dimensional space (figure 1-middle). Assuming that the content provided by *D* should be located in the right part of the zone managed by node *C* and this is the zone that will be split, user *D* is routed to *C* using the Plaxton routing, and zone *C* is split across the horizontal dimension (assuming that the previous split of *C* was, and the following split of both *C* and *D* will be performed across the vertical dimension). Finally, the recently inserted node, managing the zone *D* notifies its neighbors (i.e., the users managing zones *B* and *C*) about the insertion of a new node, and also their neighbors' lists are updated. Note that only the zone managed by user *C*, which was split and a subset of its neighbor zones (actually, only one zone managed by user *B*), are affected by the insertion of a new user *D*, whereas other zones are not affected.

Disconnections of the users are handled in a similar manner. The disconnecting user identifies one of the neighbor nodes that will takeover managing its zone, and updates other neighbor zones about the departure and the management takeover. For example, consider the user managing zone *B* disconnecting from CAN space (figure 1-right). As a result of the disconnection, the user managing zone *D* takeovers the management the zone previously managed by user *B*.

Thus, CAN provides a decentralized platform, supporting (1) dynamic space partitioning and zones allocation, (2) efficient routing algorithm, and (3) connectivity maintenance algorithm over virtual *N*-dimensional coordinate space. Note that the distributed structure of CAN is not robust against sudden departures of users, as fault-tolerance is not one of the main goals of the platform. However, CAN facilitates a decentralized self-manageable platform for content-addressable data management in a distributed environment.

4 CF over Content-Addressable Space

This work proposes a heuristic variant of the CF. It uses a content-addressable architecture for the purposes of optimizing traditional exhaustive *K*-Nearest Neighbors (KNN) search to a search among potentially similar users only. Although our algorithm is a heuristic one by nature, experimental results demonstrate that it facilitates efficient search process without hampering the accuracy of the generated predictions.

4.1 Mapping User Profiles to Content-Addressable Space

The input for the CF algorithm is a matrix of users' ratings on items, where each row (ratings vector) represents the ratings of a single user and each column represents the ratings on a single item. The total number of items (N) defines an N-dimensional space, where the coordinates range in each dimension corresponds to the range of ratings on the respective item. To handle the ratings matrix in a content-addressable manner, we map it to a CAN-like multi-dimensional space. Each rating is projected using a uniform *injective (one-to-one) mapping* onto the appropriate dimension, such that the whole vector of length N is mapped to a single point in an N-dimensional space. For example, consider a system storing the ratings of users on three different items. In such a system, the evolving CAN-like space will be a 3-dimensional cube, where the range of coordinates within every dimension corresponds to the range of possible ratings on the respective item.

As already mentioned, each user is represented in a CAN-like space by a single node whose location corresponds to the set of user's ratings and by the respective zone For example, consider a user U that rated all three items in the above 3-dimensional cube: item i_1 was rated as r_1, item i_2 as r_2, and i_3 as r_3. U will be mapped to a location (r_1,r_2,r_3) of the space and will have exactly two neighbors in each dimension. For example, in the dimension corresponding to item i_1, U will have two neighbors, $N_1=(r_1-x,r_2,r_3)$ and $N_2=(r_1+y,r_2,r_3)$, such that both N_1 and N_2 rated i_2 as r_2 and i_3 as r_3, N_1 rated i_1 below r_1, and N_2 rated it above r_1, and there is no other user that rated i_1 as r', where $r_1-x<r'<r_1$ or $r_1<r'<r_1+y$. Similarly, U will have two neighbors in the dimension corresponding to item i_2 and to item i_3. If there is no user that provided the required combination of ratings, CAN space will maintain connectivity by connecting user U to a further node, which will serve as its virtual immediate.

Note that in the evolving CAN space, the users can be dynamically inserted and removed not only during the initialization, but also during the life cycle of the system. This is explained by the observation that the above connectivity maintenance algorithm guarantees that the structure remains connected regardless of the sudden joins and disconnections of the nodes. Nevertheless, CAN spaces can barely manage insertions of new items, as the dimension of the space should remain fixed. Thus, the proposed heuristic search (that will be discussed in the following sub-section) is applicable only over a stable matrix of users' ratings, where no new items are inserted.

Deciding on the zones split policy affects the evolving structure of the ratings vectors. In our implementation, we used the above mentioned ordered splitting policy. This policy may be sub-optimal in terms of the number of neighbor zones, resulting in a less efficient algorithm, i.e., more comparisons or retrieving less similar neighbors. However, our experiments demonstrate that even this simple policy considerably increases the efficiency of the proposed K-Nearest Neighbors (KNN) search, in comparison with the traditional exhaustive search. Evaluating other splitting policies is beyond the scope of this work.

In addition to the guaranteed connectivity, content-addressable space *inherently clusters* similar users, such that the distance between two similar users (in our case, according to the MSD similarity metric) is lower than the distance between two arbitrary users. This is achieved due to the use of an injective mapping of the ratings vector to the multi-dimensional CAN-like space, which preserves the users' similarity while mapping the ratings vectors to the numeric coordinates in the space. The following subsection shows a use of the above inherent clustering property for the purposes of developing fast heuristic variant of the KNN search.

4.2 Heuristic Nearest-Neighbors Search

The Neighborhood Formation stage of the CF over the evolving *N*-dimensional space can be schematically described as a heuristically expanding breadth-first search. The algorithm for retrieving *K*-Nearest Neighbors of a user *x* is briefly explained by the following pseudo-code. The code uses two lists of size *K*: (1) *CANDIDATES* – list of candidates for being one of the *K*-nearest neighbors, and (2) *NEIGHBORS* – list of real *K*-nearest neighbors. In principle, the algorithm needs the *CANDIDATES* list only, as the *NEIGHBORS* list only increases during the execution of the algorithm until it reaches its maximal length and contains the real *K*-nearest neighbors. For the sake of clarity, we show an algorithm that uses two lists instead of only one.

```
K_Nearest_Neighbors (user x)
(1)  let NEIGHBORS and CANDIDATES be empty lists, each of size K
(2)  let Z be the zone, where x would be mapped in the CAN space
(3)  foreach u∈ (Zℏ neighbors(Z))
(4)          compute distance(x,u)
(5)          insert  u  into  CANDIDATES,  s.t.  CANDIDATES  is  sorted
             according to the values of distances(x,u)
(6)  for i=1 to K
(7)          choose v with smallest distance(x,v) from CANDIDATES
(8)          for each w∈ neighbors(v) with unknown distance(x,w)
(9)                  compute distance(x,w)
(10)                 insert w into CANDIDATES, s.t. it remains sorted
                     according  to distance(x,v)
(11)         move v from CANDIDATES to NEIGHBORS
(12) return NEIGHBORS
```

Initially, the algorithm pretends to map the active user *x* to its location in the *N*-dimensional space (step *2*). Next, the algorithm identifies the zone *x* is mapped to, and its neighbors, i.e., users managing the neighbor zones (step *3*). For each of these zones, the degree of similarity, i.e., the distance between *x* and the relevant user, is computed (step *4*). Then, the neighbor users are inserted into the *CANDIDATES* list such that the whole list of candidates users is sorted according to the distances of the users from the active user *x* (steps *4* and *5*). Afterwards, the algorithm iteratively (1) selects *v*, the nearest neighbor stored in the *CANDIDATES* list (step *7*), (2) identifies the neighbors of *v* that are not in the *CANDIDATES* list yet, computes their distances from *x*, and inserts them into the *CANDIDATES*, while keeping the list sorted (steps *8*, *9*, and *10*), and (3) removes *v* from the *CANDIDATES* list and inserts it into the *NEIGHBORS* list. Finally, the algorithm returns the *NEIGHBORS* list (step *12*).

Consider an example execution of the KNN search as illustrated in figure 2. The initial structure of *2*-dimensional space is depicted in figure 2a. Nine users, from *a* to *i*, are inserted into the space and manage the respective zones. Note that also this figure shows only the zones managed by the users, whereas the nodes representing the users are not shown. Assume that the active user is mapped to the zone managed by user *e*. Thus, *e* and its neighbors, i.e., users managing zones *c*, *d*, *f*, and *i*, are the first candidates for being the nearest neighbors and they are inserted into the *CANDIDATES* list. Assume that the user managing zone *e* is the closest one. It is moved from the *CANDIDATES* list to the *NEIGHBORS* list (figure 2a). Since all the neighbors of *e* are already known, the next closest neighbor is chosen among its neighbors. Assume that the next closest neighbor is the user managing zone *f*. It is moved from the *CANDIDATES* list to the *NEIGHBORS* list, and its only new neighbor, the user managing zone *g*, is inserted into the *CANDIDATES* list (figure 2b). The next closest neighbor is the user managing zone *c*, inserting the user managing zone *b* into the *CANDIDATES* list (figure 2c). Assume that the next closest neighbor is the user managing zone *g* (not an immediate neighbor of *e*). As a result, the user managing zone *h* is inserted into the *CANDIDATES* list (figure 2d). This process is repeated until the *NEIGHBORS* list contains *K*-Nearest Neighbors.

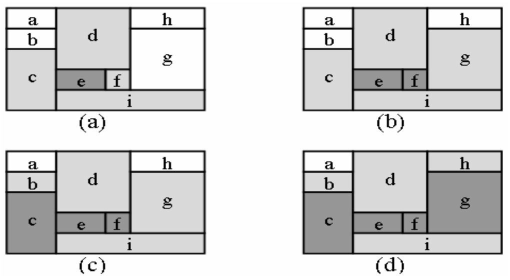

Fig. 2. Stages of the KNN search over 2-Dimensional CAN space (zones managed by users from the CANDIDATES are in a light and from the NEIGHBORS – in a dark gray)

The proposed algorithm reduces the computational effort required by the Similarity Computation and the Neighborhood Formation stages, in comparison with the traditional CF algorithm, where an active user is compared with all the available users. Conversely, the proposed heuristic algorithm compares the active users with potentially similar users only, located in close vicinity to the active user. Since every user in the *N*-dimensional space continuously maintains an updated list of its immediate neighbors, any neighbor of a given user is accessed through a single network hop. This is true regardless of the physical (geographical) and logical (similarity) distances between the neighbors. Thus, the algorithm will also work in sparse spaces, where the distance between neighbors in the underlying network might be very high.

4.3 Heuristic Completions of User Profiles

In the former sections, we assumed that the user's ratings were represented as a *complete vector*, i.e., explicit ratings on all the items are available. Thus the mapping of the user's ratings vectors to the underlying content-addressable space is straightforward. However, this assumption is unachievable in most real-life applications and scenarios, where an average user rates only a portion of the available items. This raises a need for developing a mapping mechanism capable of mapping *incomplete vectors*, where a subset of the ratings is missing, to the content-addressable space. In this sub-section we propose three mappings to handle this task. However, instead of developing a new mapping of incomplete vectors to the content-addressable space, we propose to convert the incomplete vectors to complete ones by heuristically filling-in the missing ratings in the incomplete vectors [2]. Thus, the proposed completion heuristics are designed to re-use the above injective mapping of complete vectors, while employing it on the modified vectors with heuristically filled-in ratings.

As the completion heuristics are not the main focus of the current work, we suffice with three relatively simple heuristics that demonstrate the applicability of the proposed vectors' completion. The heuristics are as follows:

- *User-average* – the missing rating on an item in the user's vector is substituted with the average of the real ratings, explicitly provided by this user.
- *Item-average* – the missing rating on an item in the user's vector is substituted with the average of the real ratings, explicitly provided by the other users on this item.
- *Conditional* – integrates both the user-average and the item-average heuristics and decides in a run-time regarding the specific completion heuristic to be used according to a certain predefined condition.

Clearly, the *user-average* heuristic can be considered as an accurate personalized completion heuristic, as the missing ratings are substituted with a value, produced by the real ratings of the given user. Thus, it reflects the real preferences and tendencies of the user, such as over- or under-rating of items, natural intensity of expressions and so forth. Conversely, the *item-average* heuristic can be considered as the most accurate non-personalized completion heuristic, as the missing ratings are substituted with a value, produced by numerous real ratings on the given item. As such, it reflects a general (and relatively reliable) opinion of many other users on the item. We conjecture that the *user-average* heuristic is preferable when the knowledge about the user's preferences is reliable, i.e., the number of ratings explicitly provided by the user is relatively high. On the other hand, when the number of user's explicit ratings is low, the *item-average* heuristic will exploit other users' ratings for filling-in the missing rating and it should be preferred. Based on these considerations, we defined another *conditional* heuristic, which will autonomously decide which of the above completion heuristics should be exploited for filling-in the missing ratings of every user.

In summary, each of these heuristics allows the filling-in of the missing ratings, converting the incomplete vectors to the complete ones, and then mapping them to the content-addressable space using the above mentioned injective mapping mechanism.

5 Experimental Evaluation

In the experimental part of our work we used the Jester dataset of jokes' ratings [6]. Jester is a Web-based joke Recommender System, containing *4.1* millions of ratings (on a continuous scale from *–10.00* to *+10.00*) of *73,421* users on *100* jokes. A significant portion of the users rated all the jokes, so the Jester dataset is relatively dense. Overall, approximately *56%* of all the possible ratings in the matrix are present. For the complete vectors experiments, we selected a subset of *14,192* users that rated all *100* jokes, producing a matrix, where every value corresponds to a real rating, explicitly provided by a user. The average rating of a joke is *0.807*, and the standard deviation of the ratings in the matrix is *4.267*. We implemented a centralized simulation of a *100*-dimensional CAN space and inserted the above *14,192* users into the space. Insertions of the users into the space were done using the ordered splitting policy.

5.1 Scalability of the Search

These experiments were designed to evaluate the scalability of the proposed heuristic variant of the KNN search. The efficiency of CAN-based KNN is measured by the number of comparisons performed by the Neighborhood Formation stage of the CF.

In this experiment we measured number of comparisons during the Neighborhood Formation stage. For this, we gradually increased the number of users inserted into the system from *M=1,000* to *M=14,000*. For each *M*, we computed the number of comparisons performed in the traditional exhaustive KNN search and in CAN-based heuristic variant of KNN. Both searches were aimed at retrieving *K=5* nearest neighbors. For each value of *M*, the experiments were repeated *1,000* times for different active users. The results are shown on Figure 3. The horizontal axis stands for *M*, the number of users inserted into the system, and the vertical axis reflects the average number of comparisons during a single KNN search, for both exhaustive and heuristic searches.

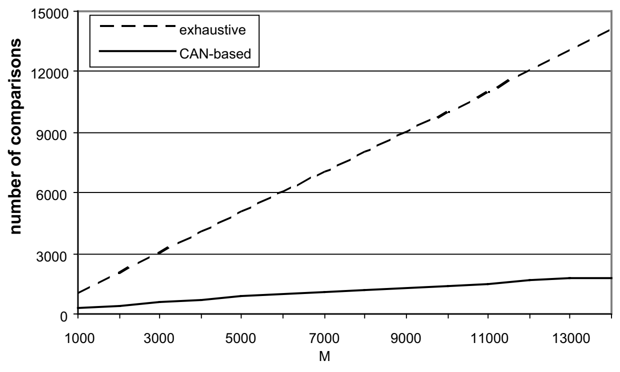

Fig. 3. Average number of comparisons vs. the number of users inserted

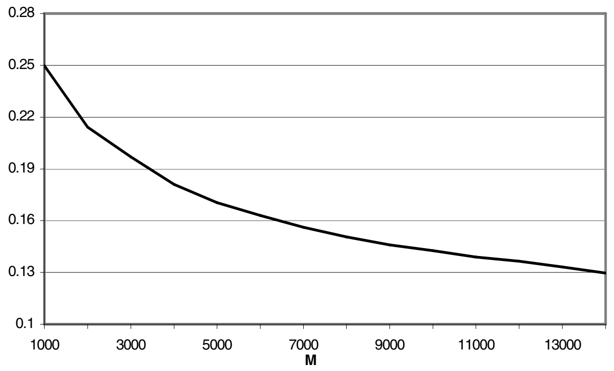

Fig. 4. Ratio between the number of comparisons vs. the number of users inserted

As expected, the number of comparisons in CAN-based KNN is significantly lower than in traditional KNN and it grows at a logarithmic-like manner with the number of users. This is explained by the fact that in CAN-based KNN the active user is compared only with a subset of highly similar users (located in close vicinity in a content-addressable space), whereas in traditional KNN it is exhaustively compared with all the available users. To better understand the scalability of the proposed approach, we computed the ratio between the number of comparisons in CAN-based KNN and the number of comparisons in the exhaustive KNN. This ratio was computed for different values of *M* and the results are shown on Figure 4. It can be seen that the ratio steadily decreases with *M*. This allows us to conclude that the proposed algorithm is applicable in large-scale systems with high number of users and items, e.g., on the Web.

The second experiment was designed to evaluate the scalability of CAN-based KNN with the number of nearest neighbors (*K*) to be retrieved. We gradually increased the value of *K* from *K=1* to *K=50*. For each value of *K*, we measured the number of comparisons needed to retrieve *K* nearest neighbors for *M=1,000*, *2,000*, *4,000*, *8,000*, and *14,000* users. For each value of *M* and *K*, the experiments were repeated *1,000* times for different active users. The number of comparisons as a function of *K* for the above values of *M* is shown on Figure 5. The horizontal axis stands for *K*, the number of nearest neighbors to be retrieved, whereas the vertical reflects the average number of comparisons during the KNN search.

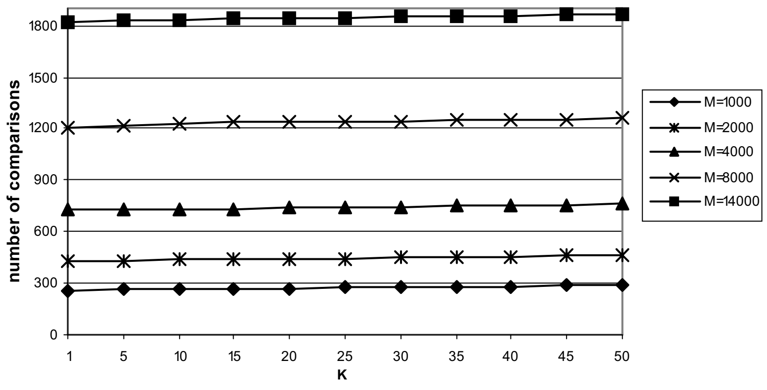

Fig. 5. Average number of comparisons vs. the number of retrieved neighbors

As can be seen, the number of comparisons in CAN-based KNN remains roughly unchanged when *K* increases. This is explained by the observation that most of the KNN users are located in close vicinity to the active user. Thus, the similar users are discovered in the early stages of the KNN search, while further expansions contribute very few new similar users. Both experiments show good scalability of CAN-based KNN with *K*. This means, that practical Recommender Systems can use higher values of *K*, to form moderately larger and more reliable neighborhoods, and generate more accurate predictions with only a very minor computational overhead.

5.2 Accuracy of the Search

The following experiments were designed to evaluate the accuracy of the results obtained by the proposed heuristic variant of KNN search. In the first experiment we compared the sets of users, i.e., the neighborhoods, retrieved by the traditional (exhaustive) KNN and by the CAN-based variant of KNN.

Let us denote by KNN_e the set of users retrieved by the traditional exhaustive KNN search and by KNN_h the set of users retrieved by the CAN-based heuristic variant of KNN. Since the CAN-based KNN is a heuristic approach, a sub-optimal structure of zones may lead to a situation, where $KNN_e \neq KNN_h$, i.e., the heuristic search retrieves only a subset of the real K nearest neighbors. As the collaborative predictions are generated by aggregating the ratings of similar users, identifying the set of most similar users is essential for generating accurate predictions.

To evaluate the accuracy of the proposed heuristic KNN search, we adapt the traditional Information Retrieval metric of precision [18]. In fact, the computed accuracy metric is rather *precision@K*, since the overall search procedure is limited to K most similar users only. However, this metric also provides some indication about the recall of the search, as it can be considered as the recall of the search for a limited number of the most similar users to be retrieved. For the sake of clarity, this metric is referred to in the paper as *precision*. The precision is computed by:

$$precision = \frac{|KNN_e \cap KNN_h|}{|KNN_e|} = \frac{|KNN_e \cap KNN_h|}{K}$$

The cardinality of the KNN_e set was $K=10$, while the cardinality of the KNN_h set was gradually increased from $K'=1$ to $K'=100$. The precision was computed for $M=1{,}000$, $2{,}000$, $4{,}000$, $8{,}000$ and $14{,}000$ users inserted into the system. For each value of M and K', the experiments were repeated $1{,}000$ times for different active users. Figure 6 shows the precision as a function of K' for the above values of M. The horizontal axis stands for M, the number of users inserted into the system, whereas the vertical reflects the average precision of the heuristic KNN search.

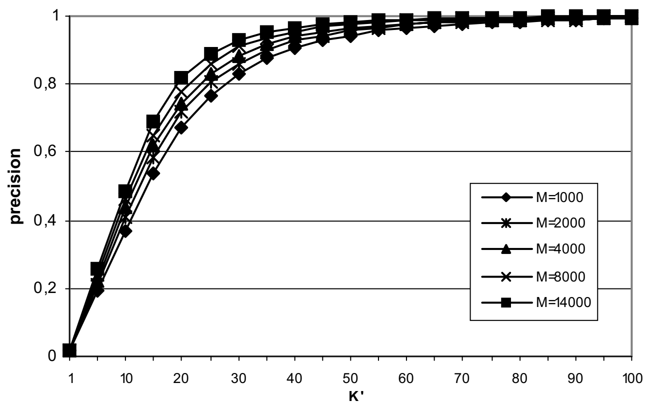

Fig. 6. Precision of CAN-based KNN

As can be seen, the curves behave similarly and the accuracy increases with *K'*, such that for *K'>50* it is over *0.9* for all the given values of *M*. Previous experiments presented in previous sub-section show that the algorithm is highly scalable with *K*. Thus, retrieving a larger set of users (i.e., higher values of *K'*) leads to a minor increase in the computational overhead. Hence, it is feasible to moderately increase the number of neighbors retrieved by CAN-based search in order to achieve a higher accuracy and generate better predictions.

Since the precision of the heuristic CAN-based KNN search may seem low for small values of *K'*, we conducted another two experiments, aimed at evaluating the quality of the neighborhood retrieved by the heuristic search. In the first, this was done by computing the average similarity between the nearest neighbors retrieved by the heuristic search and the active user. The computed average similarity was compared to the average similarity of neighborhood retrieved by the traditional search.

In the experiment, we gradually increased the number of users inserted into the system from *M=1,000* to *M=14,000*. For each value of *M*, we compared the average similarity of heuristically retrieved neighbors with the average similarity of exhaustively retrieved neighbors for *K=K'=10*. For each value of *M*, the above experiments were repeated *1,000* times for different active users. The results of the experiment are shown on Figure 7 (discussed after Figure 8). The horizontal axis stands for the number of users inserted into the system, whereas the vertical reflects the average similarity between the users in the *KNN* set and the active user for both searches.

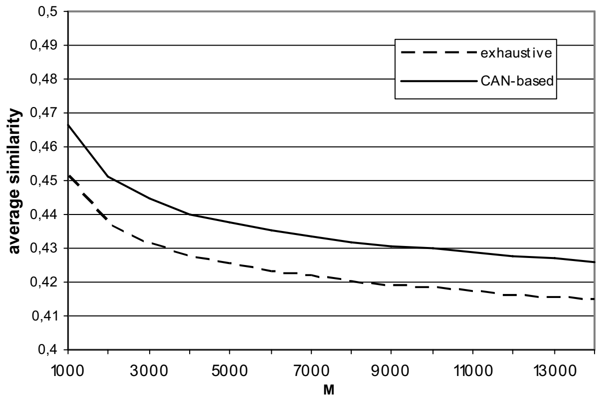

Fig. 7. Average similarity vs. the number of users inserted

The second experiment was designed to evaluate the quality of the heuristically retrieved neighborhood by comparing the accuracy of the generated predictions. The final goal of the KNN search is to retrieve a set of the most similar users, whose ratings will be aggregated when generating the predictions. Thus, we generated the predictions using both exhaustively and heuristically retrieved sets of *K*-Nearest Neighbors and evaluated the accuracy of the predictions using well-known Mean Average Error (MAE) metric [9]:

$$MAE = \frac{\sum_{i=1}^{N} | p_i - r_i |}{N}$$

where N is the number of predictions, p_i and r_i are the predicted and real ratings i.

Also in this experiment the number of users inserted into the system was gradually increased from *M=1,000* to *M=14,000*. For each value of *M*, the experiment was repeated *1,000* times for various, randomly chosen active users. For each active user chosen, the following operations were conducted: (1) a single randomly selected rating in the user's profile was hidden and served as a rating to be predicted, while the remaining *all-but-one* ratings served as the user's profile, (2) basing on the all-but-one user's profile, the set of *K=K'=10* nearest neighbors was retrieved using both traditional exhaustive and heuristic retrievals, (3) predictions were generated using both heuristically and exhaustively retrieved neighborhoods, and (4) the MAE error of the generated predictions relatively to the original hidden rating was computed. The average values of the MAE computed for certain values of *M* are shown on Figure 8. The horizontal axis stands for the number of users inserted into the system, whereas the vertical reflects the MAE values for both exhaustive and heuristic searches.

The results show that the average similarity (which is actually the dissimilarity) and the MAE of the predictions decrease with *M*. This is explained by the observation that the probability of discovering a similar user increases with the number of users inserted into the system. Thus, the average dissimilarity of the retrieved *K*-Nearest Neighbors decreases with *M*, while the accuracy of the generated predictions increases, and the MAE decreases as well.

Although both the similarity and the MAE of CAN-based heuristic search are higher (i.e., the retrieved neighbors are more dissimilar and the accuracy is actually lower), the curves are very close and the results are quite similar. Average deviation of the similarities is *2.93%* and of the MAEs is only *0.38%*. Note that the average deviation of the MAE is significantly lower than the average deviation of the similarities, as the generated predictions are barely affected by the changes in the retrieved neighborhoods. These experiments allow us to conclude that the proposed heuristic algorithm succeeds in both retrieving similar neighborhoods and generating accurate predictions.

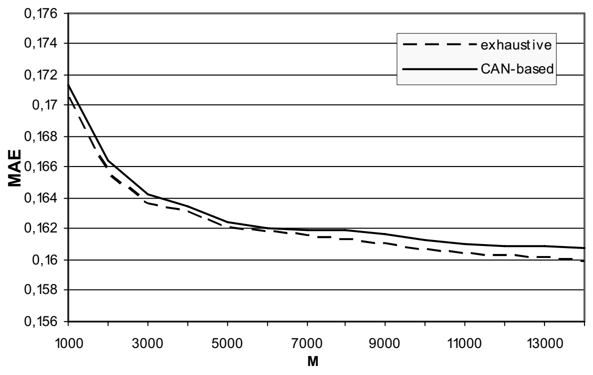

Fig. 8. Mean Average Error of the predictions vs. the number of users inserted

5.3 Inherent Clustering

One of the basic assumptions, that allows us to limit the heuristic search to users, located in close vicinity to the active user, is the *inherent clustering*. That means that the distance between two similar users is lower than the distance between two arbitrary users. Thus, the following experiment was designed to verify the property of inherent clustering in the underlying content-addressable space.

For this, we computed the average and the standard deviation of the similarity of the users located *R=1*, *2*, and *3* routing hops from the active user. The experiments were conducted for *M=1,000*, *2,000*, *4,000*, *8,000* and *14,000* users inserted into the system. For each value of *M*, the experiments were repeated *1,000* times for different orders of inserting the users into the system and for different active users. Figure 9 shows the average similarity and the standard deviation as a function of *R* for the above values of *M*. The horizontal axis stands for *M*, the number of users inserted into the system, whereas the vertical reflects the average and the standard deviation of the similarity of the retrieved users within a given number of hops from the active user.

It can be seen that for any value of *M* the similarity increases with *R*. This means that the similarity of users, located close to the active user is higher than the similarity of those located far. Thus, this experiment verifies our assumption on the clustering in content-addressable space. For any *R*, the average similarity and the standard deviation steadily decrease with *M*. This is explained by the fact that higher number of users leads to a better organization of zones, where zones managed by more similar users *block* the zones managed by dissimilar users. Thus, the average similarity (and the standard deviation) of users located within a given number of hops decreases with *R*.

Moreover, this experiment demonstrates the stability of the proposed CAN-based structure of users. This experiment was repeated *1,000* times, for different random orders of inserting the users into the system. Low values of the standard deviation, and the steady decrease of it with the number of users in the system, show that the inherent clustering holds regardless of the different types of organization of the CAN zones, imposed by the different orders of inserting the users. Thus, the proposed KNN search will succeed to retrieve accurate neighborhoods for various system usage scenarios.

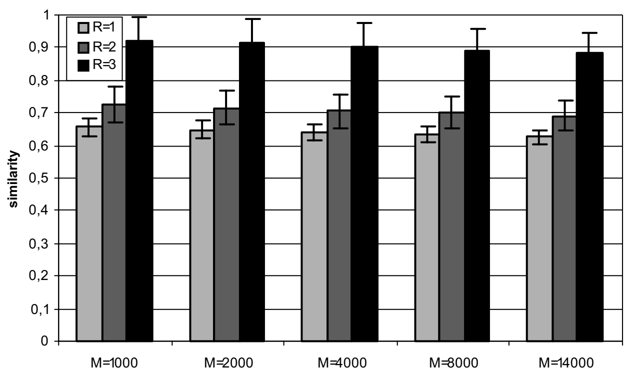

Fig. 9. Average similarity vs. number of hops from the active user

5.4 Completion Heuristics

The following experiments were designed to evaluate the proposed completion heuristics for filling-in the missing values in the incomplete ratings vectors. To run the experiment with incomplete vectors, we used the full Jester dataset [6]. In previous experiments we used a partial dataset of complete vectors, built by *14,192* users that rated all *100* jokes. In addition, the full dataset also contains the ratings of *59,229* users that rated on average *45.26* jokes. The full Jester dataset (containing both complete and incomplete vectors) was used in the completion heuristics experiments.

We implemented the *user-average* and the *item-average* completion heuristics. As for the *conditional* heuristic, the decision regarding the chosen completion heuristic was based on the number of rated items in user's ratings vector. Since in the full Jester dataset the average number of items rated by a user was *45.26*, in our implementation of the *conditional* heuristic the threshold was set to *20* items. This means that if a user rated less than *20* items, her ratings vector is not considered reliable, and the *item-average* heuristic is applied. Otherwise, the *user-average* heuristic is applied.

To evaluate the accuracy of the proposed completion heuristics, we conducted two experiments. In the first, we compared the average similarity between the active user and the *K*-Nearest Neighbors retrieved by the heuristic search and by the traditional exhaustive search. The experiment was repeated three times, for the different completion heuristics exploited before inserting the completed vectors to the underlying content-addressable space. In the experiment, we gradually increased the number of users inserted into the system from *M=5,000* to *M=50,000*. For each value of *M*, we compared the average similarity of the retrieved neighbors (using both exhaustive and heuristic retrieval techniques) for *K=K'=10*. For each value of *M*, the experiments were repeated *1,000* times for different active users. The results of the experiment are shown on Figure 10. The horizontal axis stands for *M*, the number of users inserted into the system, whereas the vertical reflects the average similarity between the users in *KNN* set and the active user for exhaustive and heuristic searches.

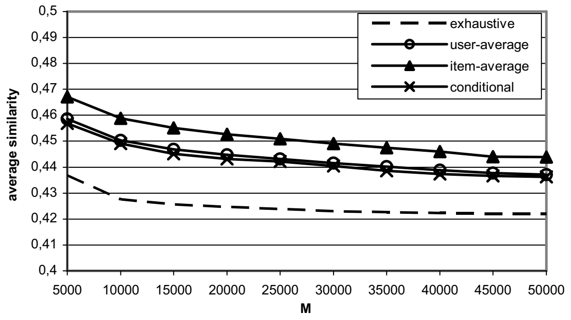

Fig. 10. Average similarity vs. the number of users inserted

The curves show, that similarly to the accuracy results in previous sub-sections, the average similarity (i.e., dissimilarity) of the retrieved KNN users decreases with *M*, the number of users inserted into the system. Comparison of the proposed completion heuristics yields that the personalized *user-average* heuristic outperforms the non-personalized *item-average* heuristic. Average similarity deviation of the KNN set exploiting the *user-average* heuristic from the exhaustively retrieved KNN is *4.43%*, while the similarity deviation of the *item-average* KNN set is *6.21%*. Since the *conditional* heuristic is a smarter combination of the above heuristics, it slightly outperforms the *user-average* heuristic as well, and for it the average similarity deviation from the exhaustively retrieved KNN set is *4.11%*.

Since the goal of the Collaborative Filtering is to generate predictions, the second experiment was designed to evaluate the quality of the completion heuristics by comparing the accuracy of the generated predictions. Hence, we generated the predictions using both exhaustively and heuristically retrieved sets of *K*-Nearest Neighbors and evaluated the accuracy of the predictions using the MAE metric. In the experiment the number of users inserted into the system was gradually increased from *M=5,000* to *M=50,000*. For each value of *M*, the experiment was repeated *1,000* times for various active users.

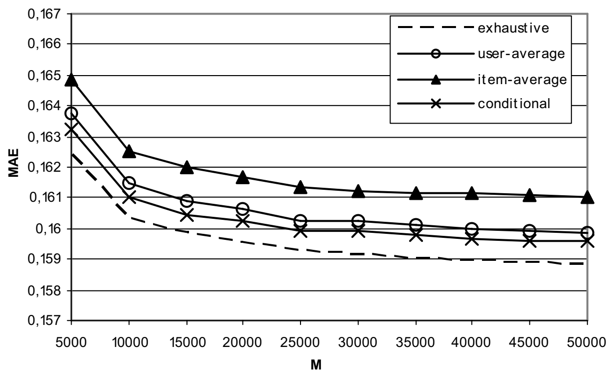

Fig. 11. Mean Average Error of the predictions vs. the number of users inserted

The experimental setting was similar to the previous sub-sections: the sets of *K=K'=10* nearest neighbors was retrieved using both exhaustive and heuristic retrievals, the predictions were generated using both neighborhoods, and the MAE of the generated predictions relatively to the original rating was computed. The average values of the MAE are shown on Figure 11. The horizontal axis stands for the number of users inserted into the system, while the vertical reflects the MAE values for both exhaustive and heuristic searches. Note that the heuristic retrieval was conducted three times, according to the completion heuristics being exploited.

Similarly to previous results, this experiment shows that the MAE of the prediction decreases with *M*, the number of users inserted into the system. Comparison of the proposed completion heuristics yields that the accuracy of the predictions using personalized *user-average* heuristic is better than of the non-personalized *item-average* heuristic. However, for both heuristics the increase of the MAE is minor: for the *user-average* heuristic it is *0.69%*, whereas for the *item-average* heuristic it is *1.37%*. As can be seen, also in this experiment the *conditional* heuristic outperforms both of them, as for the *conditional* heuristic the increase of the MAE is *0.46%*. Hence, out of the proposed completion heuristics, the *conditional* heuristic retrieves the most similar KNN set and generates the most accurate prediction. This allows us to conclude that this heuristic should be used for converting the incomplete vectors to complete ones, and leads to future research on developing more accurate completion heuristics.

6 Conclusions and Future Work

One of the major drawbacks of the state-of-the-art CF implementations is their high computational complexity, which grows linearly both with the number of users and items in the system. In this work we proposed to heuristically decrease the required computational effort by implementing the CF over content-addressable CAN-like *N*-dimensional space.

Experiments conducted over the Jester dataset of joke ratings show that the proposed heuristic algorithm outperforms the traditional exhaustive KNN search as the computational overheads are significantly decreased, while the accuracy remains similar. Our algorithm decreases the number of required comparisons, such that the ratio between the numbers of comparisons steadily decreases with the number of users. For example, for *14,000* users the number of comparisons was decreased by *87%*. Other experiment shows that the number of comparisons roughly remains unchanged when *K* increases. This allows us to increase the number of nearest neighbors retrieved (and to improve the accuracy of the predictions) with a minor computational overhead.

In the accuracy experiments we qualitatively compared the neighborhoods retrieved and the predictions generated by the CAN-based heuristic and by the traditional exhaustive KNN searches. The retrieved neighborhoods were similar and the predictions were close, which indicates a good accuracy of the proposed algorithm. In summary, comparing the proposed heuristic search with traditional exhaustive search shows that our algorithm achieves high accuracy, while significantly decreasing the required computational effort. Another experiment was aimed at validating the inherent clustering property of content-addressable spaces. The results showed that this property holds in the CAN-like space, as the dissimilarity of users, located in a certain number of network hops from the active user increased with the number of network hops. The experiments also showed that the inherent clustering property holds regardless of the number of users inserted into the system and the order of their insertion.

Finally, the last experiment was aimed at comparing three simple heuristic for converting the incomplete vectors to complete ones by filling-in the missing ratings. The experimental results showed that the heuristic, which conditionally integrates two other heuristics, outperforms them both in terms of the retrieved neighborhoods' similarity and of the generated predictions' accuracy. Comparing the MAE of the predictions generated by the complete and heuristically completed vectors yields that the accuracy of the predictions generated by the complete vectors is slightly better. This conclusion is reasonable, since the proposed completion heuristics insert some extent of noise into the original ratings. However, the increase in the MAE is minor, allowing us to conclude that the achieved computational optimization is preferential than the minor noises in the predictions caused by the artificial ratings inserted by the completion heuristics.

In this work, we inherently assumed that the system assigns equal relative weights to the ratings on each item. However, this assumption is not true in many real-life personalization applications. For example, this assumption might be false in a situation, where different criteria affect differently on the similarity values, e.g., when the similarity values between the items are known. Developing a weighted prediction algorithm will result in a more accurate Recommender System.

Also, we assumed that either the user's ratings on the items are available or they can be easily filled-in using one of the proposed simple completion heuristics. However, some real-life scenarios, this completion is hard to achieve, since the matrix is very sparse (e.g., density of *2-3%* in typical Collaborative Filtering datasets such as [9] and [10]) and the substitution of the missing values may require exploiting more intelligent techniques. In the future, we plan to study the use of various completion heuristics, exploiting statistical and Machine Learning techniques.

In addition to decreasing the computational effort, the proposed algorithm can naturally be extended to distribute it among multiple users. In traditional centralized implementations of the CF, the Similarity Computation and the Neighborhood Formation stages are performed in a single central location. However, as the underlying CAN platform is originally distributed Peer-to-Peer platform, it inherently allows distributed and fully decentralized storage of the ratings matrix. In the future, we plan to implement a distributed variant of the algorithm and to investigate the distribution issues.

The current work is limited to the Mean Squared Difference (MSD) similarity metric, since the injective mapping to a multi-dimensional CAN-like space inherently supports it. However, for other metrics, such as Cosine Similarity or Pearson Correlation, CAN space might be inappropriate and new types of topologies and respective mappings should be developed. We plan to study other metrics and to produce a general framework for efficient heuristic Collaborative Filtering.

References

1. Aguzzoli, S., Avesani, P., Massa, P.: Collaborative Case-Based Recommender System. In: Proceedings of the ECCBR Conference (1997)
2. Bogaerts, S., Leake, D.: Facilitating CBR for Incompletely-Described Cases: Distance Metrics for Partial Problem Descriptions. In: Funk, P., González Calero, P.A. (eds.) ECCBR 2004. LNCS (LNAI), vol. 3155, Springer, Heidelberg (2004)
3. Breese, J., Heckerman, D., Kadie, C.: Empirical Analysis of Predictive Algorithms for Collaborative Filtering. In: Proceedings of the UAI Conference (1998)
4. Burke, R.: Hybrid Recommender Systems: Survey and Experiments. User Modeling and User-Adapted Interaction 12(4), 331–370 (2002)
5. Chee, S.H.S., Han, J., Wang, K.: RecTree: An Efficient Collaborative Filtering Method. In: Kambayashi, Y., Winiwarter, W., Arikawa, M. (eds.) DaWaK 2001. LNCS, vol. 2114, Springer, Heidelberg (2001)
6. Goldberg, K., Roeder, T., Gupta, D., Perkins, C.: Eigentaste: A Constant Time Collaborative Filtering Algorithm. Information Retrieval Journal 4(2) (2001)
7. Good, N., Schafer, J.B., Konstan, J.A., Borchers, A., Sarwar, B., Herlocker, J., Riedl, J.: Combining Collaborative Filtering with Personal Agents for Better Recommendations. In: Proceedings of the AAAI Conference (1999)
8. Han, P., Xie, B., Yang, F., Shen, R.: A Scalable P2P Recommender System Based on Distributed Collaborative Filtering. Expert Systems with Applications Journal 27(2) (2004)
9. Herlocker, J.L., Konstan, J.A., Borchers, A., Riedl, J.: An Algorithmic Framework for Performing Collaborative Filtering. In: Proceedings of the SIGIR Conference (1999)
10. McJones, P.: Eachmovie Collaborative Filtering Data Set (1997), http://research.compaq.com/SRC/eachmovie/

11. Miller, B.N., Konstan, J.A., Riedl, J.: PocketLens: Toward a Personal Recommender System. ACM Transactions on Information Systems 22(3) (2004)
12. Morita, M., Shinoda, Y.: Information Filtering Based on User Behavior Analysis and Best Match Retrieval. In: Proceedings of the SIGIR Conference (1994)
13. Pennock, D.M., Horvitz, E., Giles, C.L.: Social Choice Theory and Recommender Systems: Analysis of the Axiomatic Foundations of Collaborative Filtering. In: Proceedings of the AAAI Conference (2000)
14. Plaxton, C., Rajaraman, R., Richa, A.: Accessing Nearby Copies of Replicated Objects in a Distributed Environment. In: Proceedings of the ACM SPAA Conference (1997)
15. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S.: A Scalable Content-Addressable Network. In: Proceedings of the SIGCOMM Conference (2001)
16. Resnick, P., Varian, H.R.: Recommender Systems. Communications of the ACM 40(3) (1997)
17. Ricci, F., Venturini, A., Cavada, D., Mirzadeh, N., Blaas, D., Nones, M.: Product Recommendation with Interactive Query Management and Twofold Similarity. In: Ashley, K.D., Bridge, D.G. (eds.) ICCBR 2003. LNCS, vol. 2689, Springer, Heidelberg (2003)
18. Salton, G., McGill, M.: Introduction to Modern Information Retrieval. McGraw-Hill Pubslishers, New York (1983)
19. Sarwar, B., Karypis, G., Konstan, J., Riedl, J.: Analysis of Recommendation Algorithms for E-Commerce. In: Proceedings of the EC Conference (2000)
20. Sarwar, B.M., Konstan, J.A., Riedl, J.: Distributed Recommender Systems: New Opportunities for Internet Commerce. In: Internet Commerce and Software Agents: Cases, Technologies and Opportunities, Idea Group Publishers, USA (2001)
21. Shardanand, U., Maes, P.: Social Information Filtering: Algorithms for Automating "Word of Mouth". In: Proceedings of the CHI Conference (1995)
22. Tveit, A.: Peer-to-Peer Based Recommendations for Mobile Commerce. In: Proceedings of the WMC Workshop (2001)