Knowledge-Based Systems 51 (2013) 27–34

Contents lists available at SciVerse ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

A similarity metric designed to speed up, using hardware, the recommender systems *k*-nearest neighbors algorithm

Jesús Bobadilla [a,*], Fernando Ortega [a], Antonio Hernando [a], Guillermo Glez-de-Rivera [b]

[a] *Universidad Politécnica of Madrid & FilmAffinity.com research, Spain*
[b] *Universidad Autónoma de Madrid, Spain*

ARTICLE INFO

Article history:
Received 4 February 2013
Received in revised form 24 May 2013
Accepted 14 June 2013
Available online 16 July 2013

Keywords:
Similarity measure
Metric
Collaborative filtering
Memory based
Recommender system

ABSTRACT

A significant number of recommender systems utilize the *k*-nearest neighbor (*k*NN) algorithm as the collaborative filtering core. This algorithm is simple; it utilizes updated data and facilitates the explanations of recommendations. Its greatest inconveniences are the amount of execution time that is required and the non-scalable nature of the algorithm. The algorithm is based on the repetitive execution of the selected similarity metric. In this paper, an innovative similarity metric is presented: *HwSimilarity*. This metric attains high-quality recommendations that are similar to those provided by the best existing metrics and can be processed by employing low-cost hardware circuits. This paper examines the key design concepts and recommendation-quality results of the metric. The hardware design, cost of implementation, and improvements achieved during execution are also explored.

1. Introduction

Recommender Systems (RS) provide a relevant tool which helps to mitigate part of the information overload generated via the use of Web 2.0 applications. RS provide personalized recommendations to users about items (books, music, films, gadgets, holiday destinations, etc.) [26,20,23,4,8,24].

When the RS is solely based on the information stored in the array of votes it is called memory-based collaborative filtering [1,16,14]. A variety of Collaborative Filtering (CF) exists which obtains information from additional sources to the array of votes, such as the social relations between users or the contents of posts in blogs; in these cases (memory-based + additional information) the additional information is used to improve the quality of the recommendations, but its use is only applicable to the subset of RS where that type of additional information exists. All scientific progress in the area of memory-based CF has the virtue of being applicable in the different types of CF-based RS: pure CF and Hybrid filtering (CF + social, CF + demographic, CF + knowledge-based, etc.).

As the size of CF RS increase, the system response times also increase. The model-based CF [19] helps to reduce recommendation times, but this reduction causes the system to operate with obsolete data and requires continuous offline updates of the models. As a result, commercial RS usually utilize memory-based CF algorithms, such as the *k*-nearest neighbor (*k*NN) algorithm. This algorithm calculates a set of *k* neighbor users whose similarities are most comparable to the user for whom a recommendation is sought (the active user). The similarity between users is determined by applying metrics that act on the existing set of data (memory-based CF).

The *k*NN algorithm requires lengthy computation times because of the need to calculate the similarity of each active user comparing them with the votes cast by each of the existing users in the database. Using Netflix as an example, each recommendation created for an active user requires calculations of the similarities between the active user and approximately 480,000 other users. A maximum of 17,000 votes must be compared. A typical similarity metric is the Pearson correlation. Using this metric, the process of creating recommendations for a user requires the processing of approximately 480,000 correlations of 17,000 values.

Traditionally, the similarity metrics and measures used in RS come from those used in the statistics area or some of those used in various fields of information retrieval, such as Pearson correla-

Abbreviations: CF, Collaborative Filtering; RS, Recommender System; JMSD, Jaccard Mean Squared Differences; NN, Nearest Neighbors; *k*NN, *k* Nearest Neighbors; COR, Pearson Correlation; MAE, Mean Absolute Error; MSD, Mean Squared Differences; DFM, Deviation From Mean.

* Corresponding author. Address: Universidad Politécnica of Madrid, ctra. of Valencia Km.7, zip 28031, Madrid, Spain. Tel.: +34 91 3365133; fax: +34 91 3367522.

E-mail addresses: jesus.bobadilla@upm.es (J. Bobadilla), fernando.ortega@upm.es (F. Ortega), antonio.hernando@upm.es (A. Hernando), guillermo.gdrivera@uam.es (G. Glez-de-Rivera).

0950-7051/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.knosys.2013.06.010

tion, cosine, adjusted cosine and Spearman rank correlation [1,16]. Recent studies have shown that it is possible to improve the quality of the prediction and recommendation results [16,10,17] by using new memory-based CF similarity metrics and measures [8,9,10,11,13] specifically designed to make the most of the special feature inherent to RS and its more complex operating modes, such as high levels of sparsity [3] and cold-start situations [2].

By relegating the performance issue to marginal commentaries or brief comparisons of execution time, research in the field has focused on improvements in the accuracy, precision, and recall of predictions and recommendations. At present, a midrange modern computer is able to process the recommendation of a Netflix user in 50 ms. Real-time systems are not considered because users do not require a defined response time. In this context, improvements in the accuracy provided are more appealing than improvements in system performance.

In spite of the previous considerations, three factors contribute significant value to research aimed at improving processing times for recommendations:

1. The memory-based CF algorithms are not scalable. The processing time increases in quadratic proportion with the number of users and items (elements to recommend).
2. The number of users and items increases rapidly in commercial RS.
3. The distribution of the recommendation requests is not completely uniform: large demand spikes occur on certain days and at certain hours (for example, Sunday evenings).

The development of a mechanism that is aimed at reducing the execution times of CF RS is necessary to ensure viable processes for both current and future operations, particularly during high-demand periods and for large RS.

Although more attention has been focused on the recommendation quality of RS, there is a research topic that considers processing time for the *k*NN algorithm: the Nearest Neighbors (NN) classifiers and, more specifically, the NN graphics classifiers. There are four different techniques aimed at accelerating the classical *k*NN method:

- *Accelerate the sort operation*: In [7] they replace the sort operation with calculating the order statistics, making the *k*NN method more efficient; they also increase the method's stability. In the case of RS, the sort operation requires much less execution time than the processing of all similarity measurements; therefore, significant reductions in execution time do not exist in this portion of the algorithm. Furthermore, stability problems in the sort operation do not occur in RS because ordering is performed on sets of real values. In NN graphics classifiers, sorting algorithms order the records with equals keys in different ways.
- *Parallel approaches*: In [5] they propose a parallel Graphics Processing Units (GPUs) implementation for the *k*NN algorithm. In [6] they use the parallel implementation to accelerate brute force searching algorithms for metric-space databases. There are several factors that make the adoption of a *k*NN based CF parallel approach unappealing: sparsity and the centralized nature of RS databases, the fine granularity of the similarities metrics, and the absence of specific processors.
- *Using kd-trees and k-means clustering when using NN matching in high-dimensional spaces*: In [21,25] they use hierarchical *k*-means trees and multiple randomized *k-d* trees to provide the best performance results. In the case of RS, each ⟨user, item⟩ dimensional space datum is defined by a unique real value. Its processing does not require a matching complex; therefore, it is usually resolved by processing a statistical similarity measure, such as the Pearson correlation.
- Decreasing the number of elements that should be compared in the *k*NN algorithm: In [15] they present a technique for quickly eliminating most templates from consideration as possible neighbors. This method is applicable when the number of features is large and the type of each feature is binary. In [18] they weight all the training instances in the generalization phase; an instance having zero weight can be removed from the training set. In the case of RS, the amount of time required for filtering (or weight processing) each item or user should be similar to the amount of time required for the similarity metric, for which reason its use in CF RS is inadequate. In [22] they use Pareto dominance to perform a pre-filtering process eliminating less representative users from the *k*-neighbor selection process while retaining the most promising ones.

In a *k*NN-based CF RS, the repetitive evaluation of the similarity metric is the phase in which nearly all of the processing time is utilized. Any significant improvements in performance (time consumption) of the similarity metric will have a significant impact on any improvements in recommendation times for the entire system.

This paper introduces a similarity metric that is designed for simple and cost-effective implementation through the use of low-cost hardware. The proposed metric slightly reduces the quality of the obtained results (in comparison with the latest generation of metrics); however, its "recommendation quality/performance" relation makes it the current CF metric with the greatest advantages among large RS.

The proposed similarity measure allows to face new and exciting possibilities into the RS collaborative filtering based applications: (1) RS in which users recognize, in real time as they add new ratings, changes in the recommendations that they receive (RS push technology); (2) Direct and high performance support to RS based on binary ratings; these RS are becoming more and more popular due to the typical like/dislike GUI options of social applications running on mobile devices; and (3) RS servers devoted to process incoming recommendation requests from users of different companies services; that is, hardware circuits containing tens of thousands of parallel bit processing units, running requests from users of films, music, news, posts, etc. RS companies.

The paper is structured in the following way: Section 2 explains the method followed to design the similarity measure, Section 3 shows the hardware design and implementation approaches, Section 4 presents the experiments carried out and the results obtained. Finally, Section 5, sets out the most relevant conclusions.

2. Metric design

JMSD is a metric with a very simple formulation. Its motivation and foundations can be found in [9]. JMSD combines: (a) numerical similarity information between the 2 users compared (Mean Squared Differences "MSD") and (b) non-numerical information (structural information) of similarity between the 2 users compared (Jaccard).

JMSD provides better recommendation results than traditional metrics; in addition, its design facilitates the creation of variants that can be easily implemented via hardware circuits. For these 2 reasons, JMSD was used as a reference for the creation of the proposed similarity metric: *HwSimilarity*. JMSD combines the Jaccard and MSD metrics. *HwSimilarity* combines specific versions of these metrics: *BitJaccard* and *BitMSD*, which are adapted to the hardware implementation.

In order to define *HwSimilarity* (running on hardware circuits) from *JMSD* (defined over real numbers), we design an intermediate metric: *BitJMSD*. By means of binary data and logic operations, *Bit-*

JMSD obtains similar results to those of *JMSD*. This can be summarized according to the following:

JMSD = *Jaccard* $*$ *MSD*; *JMSD* uses *Jaccard* and *MSD* metrics, dealing with real numbers and real numbers operations.

BitJMSD = *BitJaccard* – *BitMSD*; *BitJMSD* uses *BitJaccard* and *BitMSD* metrics, dealing with Boolean data and logical operations. Finally, a natural numbers sustraction operation will be necessary.

HwSimilarity = *AND* AND *XNOR*; *HwSimilarity* uses *AND* and *XNOR* Boolean functions, dealing with Boolean data and logical operations. Finally the Boolean function AND is used to get the result.

Fig. 1 shows the evolution from *JMSD* to the proposed *HwSimilarity*. The evolution of Jaccard and *MSD* have been discussed, respectively, in Sections 2.2 and 2.3. *BitJaccard* and *BitMSD* are discussed in Section 2.4. The proposed *HwSimilarity* (sequential and parallel) is explained in Section 3. The recommendations quality improvements are showed in Section 4.1. Finally, performance (speed up) improvements are discussed in Section 4.2.

2.1. Preliminary definitions

Let I the set of the RS items. Let $V = \{\min \cdots \max\} \cup \{\bullet\}$ the range of the ratings and the lack of rating.

Let L the set of relevant ratings, and γ the relevance threshold: $L = \{\gamma \cdots \max\} | \gamma \in \{\min \cdots \max\}$.

Let R the set of ratings from users to items: $R = \{\langle u, i, r\rangle > | \forall u \in U, \forall i \in I\}$.

Metrics in this paper utilize only three possible states to define each rating $r_{u,i}$: (non-relevant: 0, relevant: 1, not voted: $\bullet$):

Let

$$R^* = \{\langle u, i, r^*\rangle\} | \forall u \forall i, \quad r^*_{u,i} = \begin{cases} 0 \iff r_{u,i} \in V - L \\ 1 \iff r_{u,i} \in L \\ \bullet \iff r_{u,i} = \bullet \end{cases} \quad (1)$$

2.2. Jaccard component

Jaccard represents the proportion of items that has been rated in common by the two users that were compared: u, v (2). *BitJaccard* does not use proportions; instead, it utilizes only the *Jaccard* numerator (3). Although this simplification greatly reduces the complexity of the required hardware, it leads to a loss of information. It is reasonable to consider the similarity of 2 users who have rated approximately 40 items each, of which 30 items were rated in common. It is not as reasonable to consider the similarity of 2 users who have rated approximately 2000 items each, of which only 30 items were rated in common.

Let I_u be the set of items voted by user u: $I_u = \{i \in I | r_{u,i} \neq \bullet\}$

$$Jaccard(u, v) = \frac{|I_u \cap I_v|}{|I_u \cup I_v|}, \quad Jaccard(u, v) \in [0 \cdots 1] \quad (2)$$

$$BitJaccard(u, v) = \sum_{i \in I} \alpha_i \quad |\alpha_i = \begin{cases} 0 \iff r^*_{u,i} = \bullet \vee r^*_{v,i} = \bullet \\ 1 \iff r^*_{u,i} \neq \bullet \wedge r^*_{v,i} \neq \bullet \end{cases}, \quad BitJaccard(u, v) \in \{0 \cdots |I|\} \quad (3)$$

In order to explain each metric and the decisions made to change the similarity measures to fit the hardware proposal, this papers provides a scenario of use showed as a running example: Given a RS with 8 items ($I = 8$), users u and v have voted as follows: u: 5 1 $\bullet$ 2 1 $\bullet$ 4 1, v: 4 2 $\bullet$ $\bullet$ 2 1 5 5. We apply the relevancy threshold value $\gamma = 4$. Fig. 2 shows the internal operations of the *Jaccard* and *BitJaccard* similarity measures.

Jaccard, on the left side of Fig. 2, needs to process the following operations: (1) to calculate, using real numbers, the quantity of times that both user u and user v have rated the same items (numerator); (2) to calculate, using real numbers, the number of times that the user u or the user v have rated any item (denominator); and (3) to calculate a division between real numbers. *BitJaccard*, on the right side of Fig. 2, simplifies the *Jaccard* process: (1) It operates just using logical values and (2) It does not need a real numbers division: Instead it uses a simple count of the logical “1” values.

To assess how disregarding the *Jaccard* denominator impacts the quality of the results, the Mean Absolute Error (MAE) of the original *JMSD* and the MAE of the *JMSD* without the *Jaccard* denominator are calculated. Fig. 3 shows the resulting loss of accuracy. The results are labeled *JMSD* and *BitJaccard* + *MSD*; *COR* refers to the *Pearson correlation* and is incorporated in the figure as a baseline value.

The percentage of average accuracy loss is 2.7%. This value is acceptable in the context of a considerable improvement in the response times of RS.

2.3. Mean squared differences component

The MSD presents a simple measurement of the average value of the discrepancy between the votes cast by the two users who are being compared (4). If the same vote had been cast for each of the items that were mutually voted for, the value of MSD will be zero (maximum similarity).

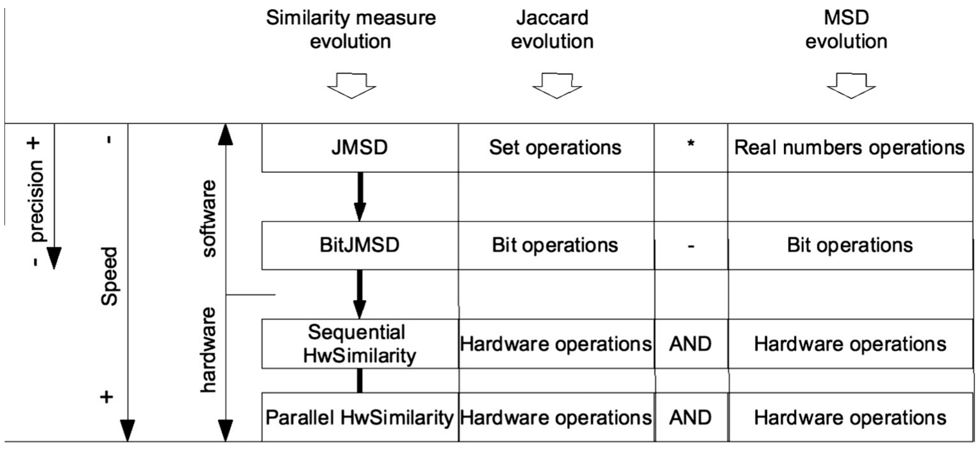

Fig. 1. Similarity measures involved in the paper and precision/performance evolution.

$$MSD(u,v) = \frac{\sum_{i \in I_u \cap I_v} (r_{u,i} - r_{v,i})^2}{|I_u \cap I_v|} \quad (4)$$

where I_u represents the items rated by the user u.

BitMSD (5) and *BitJaccard* both disregard the information provided by the denominator (in this case, the average value). *BitMSD* also disregards the numerical values of the votes; it only records the cases in which discrepancies exist for the relevance (1) of the same votes. In [9,11], an experiment is presented that examines how the "relevant/not relevant" assessment not only does not worsen the precision/recall measurements, but rather it improves them both, particularly the precision, when the number of recommendations (N) is high.

$$BitMSD(u,v) = \sum_{i \in I} \beta_i \quad |\beta_i = \begin{cases} 0 \iff \alpha_i = 0 \vee r^*_{u,i} = r^*_{v,i} \\ 1 \iff \alpha_i = 1 \wedge r^*_{u,i} \neq r^*_{v,i} \end{cases}, \quad BitMSD(u,v) \in \{0 \cdots |I|\} \quad (5)$$

Fig. 3 shows a negligible accuracy loss between the results of the original *JMSD* and the combination of the *Jaccard* with *HwDiff BitMSD*. This result offers an alternative formulation to the *JMSD* without apparent loss of quality and with a lower cost of hardware and implementation.

Following Fig. 2 running example, Fig. 4 shows the internal operations of the *MSD* and *BitMSD* similarity measures. *MSD*, on the left side of Fig. 4, needs to process the following operations: (1) to calculate, using real numbers, the squared errors between the items that both user u and user v have rated (numerator) and (2) to calculate a division between real numbers. *BitMSD*, on the right side of Fig. 4, simplifies the *MSD* process: (1) It operates just using logical values and (2) It does not need a real number division: Instead it uses a simple count of the logical "1" values.

2.4. Proposed similarity metric

Finally, the proposed metric, *BitJMSD* (6), computes the number of items that have been rated in common and with the same significance (relevant or not relevant). *BitJMSD* generates results in the range $\{0 \cdots |I|\}$; a value of 0 indicates no similarity, and a high value of the metric indicates a high similarity between the two users that are being compared (u, v). Eq. (6) is based on the following facts: (1) *BitJaccard*↑ ⇒ *BitJMSD*↓ and (2) *BitMSD*↓ ⇒ *BitJMSD*↑.

$$\boxed{BitJMSD(u,v) = BitJaccard(u,v) - BitMSD(u,v), \quad BitJMSD(u,v) \in \{0 \ldots |I|\}} \quad (6)$$

In the next section (Section 3), the running example clarifies the way the proposed similarity measure (*HwSimilarity*) removes the state • (not rated item), making possible the use of hardware circuits.

2.5. Aggregation approach

To estimate the items predictions, the set of k neighbors obtained with *BitJMSD* is utilized in the aggregation approach phase. The execution time of the aggregation approach is negligible with respect to the time required for calculating the set of neighbors; thus, its hardware implementation is not required. Nevertheless, given that *BitJMSD* operates with values of 0, 1, and •, we formulate the *Deviation From Mean* (*DFM*) aggregation approach (8) by considering the following condition:

Let $\bar{r}^*_u$ the average rating of the user u:

$$\bar{r}^*_u = \frac{|\{i \in I | r^*_{u,i} = 1\}|}{|\{i \in I | r^*_{u,i} \neq \bullet\}|} \quad (7)$$

Let $K_{u,i}$ the set of neighbors of u that have rated the item i.

Let $p^*_{u,i}$ the prediction of the item i for the user u using the range $\{0 \cdots 1\}$:

$$p^*_{u,i} = \bar{r}^*_u + \frac{\sum_{v \in K_{u,i}} BitJMSD(u,v)(r^*_{v,i} - \bar{r}^*_v)}{\sum_{v \in K_{u,i}} BitJMSD(u,v)} \quad (8)$$

Let $p_{u,i}$ the prediction of the item i for the user u using the range $\{\min \cdots \max\}$:

$$p_{u,i} = p^*_{u,i}(\max - \min) + \min \quad (9)$$

3. Hardware design and implementation

3.1. Metric definition

An RS that takes advantage of the proposed metric will allow users to vote on the items with only 2 options: positive and not positive. An operating RS can obtain these 2 values by applying a relevance threshold γ. For both of these cases, we begin with 2 packages of I bits for each user to begin the hardware processing:

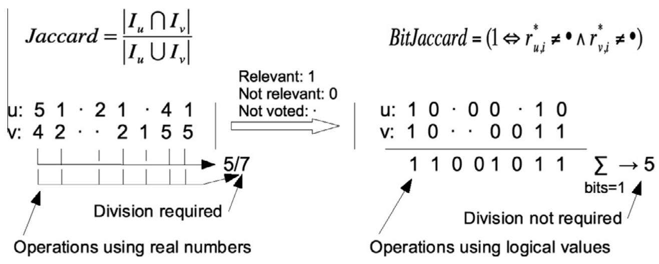

Fig. 2. Running example: *Jaccard* versus *BitJaccard* similarity measures.

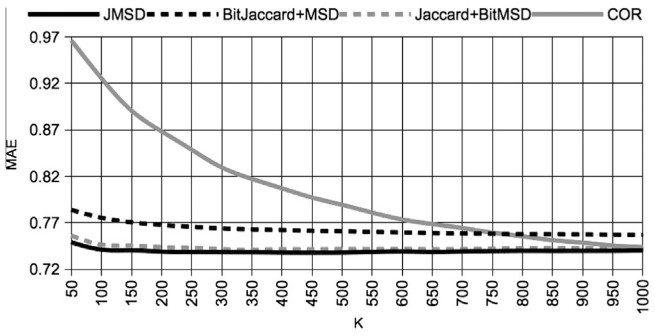

Fig. 3. Accuracy loss experienced after disregarding the *Jaccard* denominator (*BitJaccard*) and the MSD denominator (*BitMSD*) (MovieLens database has been used). y axis represents the average error of the predictions (MAE). x axis represents the number of neighbors used to process the k nearest neighbor algorithm.

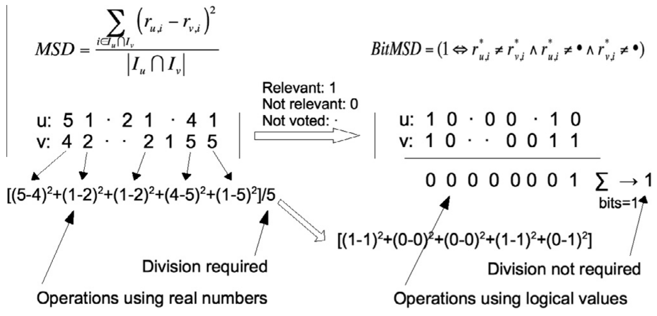

Fig. 4. Running example: *MSD* versus *BitMSD* similarity measures.

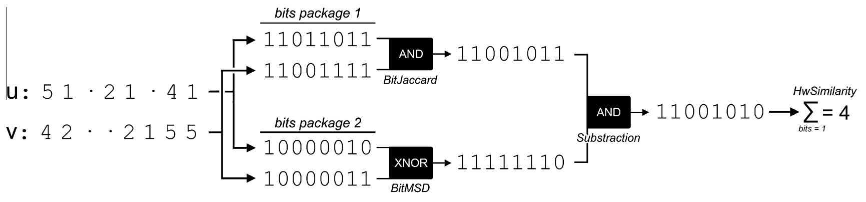

Fig. 5. Running example: Schematic of the hardware operation using the *HwSimilarity* metric.

- Package 1: items "rated/not rated" by the user.
- Package 2: items "rated as relevant/(not rated or rated as irrelevant)" by the user.

Let u_{p1} the user's u package 1, and u_{p2} the user's u package 2:

$$\boxed{HwSimilarity = \sum_{bits=1}\left(u_{p1}\cdot v_{p1}\right)\cdot\left(\overline{u_{p2}\oplus v_{p2}}\right)} \tag{10}$$

BitJaccard is implemented with the AND function applied to package 1 of u and v; *BitMSD* is implemented with the XNOR function applied to package 2 of u and v. Fig. 5 shows the running example hardware operative process. *BitJMSD* = *BitJaccard* – *BitMSD* (6). That is: Number of items both users have voted – Number of items that have been voted using a different "relevant/not relevant" assessment.

Upper side of Fig. 5 shows the *HwSimilarity* implementation of *BitJaccard*: AND operation applied to the packages_1 bits. Bottom side of Fig. 5 shows the *HwSimilarity* implementation of *BitMSD*: XNOR operation applied to the packages_2 bits. Finally, right side of Fig. 5 shows the *HwSimilarity* implementation of the *BitJMSD* subtraction: AND operation applied to the previous results.

3.2. Implementation

This section shows 2 possible hardware implementations of the metric proposed in this article. Fig. 6 shows the most economical implementation (sequential). The "Bit processing" module implements the *HwSimilarity* Eq. (10). The remaining elements are responsible for performing a sequential process: Memories 1, 2, 3, and 4 store up1, vp1, up2, and vp2, respectively. The counter C2 is responsible for calculating the sum of bits with value 1 (summation in Eq. (10)). Memory 5 stores different *HwSimilarity* results when applied to different users. Finally, the C1 counter marks each phase of the circuit, which forms a pipeline of operations.

To conduct parallel hardware implementation, the most immediate approach is to divide the set of I items into n blocks, sequentially process the I/n items of each block (copying n times the sequential implementation), and summing the I/n partial values. The inconveniences of this approach are as follows: (a) the execution time added at the last stage of summing the partial results and (b) the low scalability of a solution in which I/n partial values must be added.

To avoid the need to sum circuits of I/n entries, a parallel hardware solution is designed to process all of the similarity calculations between an active user and the remaining users in the RS database. Using this approach, the sequential implementation circuits are copied n times and the similarity between an active user and n users of the RS database is processed in parallel. Avoiding a final stage in which the results are totaled resulted in a design that is simple, scalable, fast, and economical. Fig. 7 shows the schematics of the resulting hardware.

The cost associated with the circuits in the sequential hardware implementation is approximately \$1 (without including memory, whose size depends on the volume of data in each RS database). Using the parallel version, as the value of n increases, the hardware implementation with n processing elements approaches \$$n$ (memory not included).

Both sequential and parallel approaches have been simulated using the Altium Designer's simulation engine. It uses an enhanced version of XSpice, designed to incorporate functional digital simulation capabilities. Digital devices are modeled and simulated using a descriptive language, called Digital SimCode.

4. Results

4.1. Quality results

With the aim of testing the proposed similarity measure, a series of experiments have been carried out using the databases MovieLens 1 M, Netflix and FilmAffinity.[1] In these experiments, we obtain the quality measures precision versus recall. The similarity measure proposed (*HwSimilarity*) is compared with the two reference metrics (JMSD and COR) as a baseline.

Table 1 shows the main parameters used in the experiments. Precision and recall quality measures are obtained using values of N from 2 to 20 (N represents the number of recommendations made). The relevancy threshold (θ) necessary for considering a recommendation as relevant is 5 in Movielens and Netflix (where ratings are represented from 1 to 5). θ was set to 9 using FilmAffinity (where ratings are represented from 1 to 10). The number of neighbors (k) was set to 350 in the larger databases (Netflix and FilmAffinity), whereas $k = 300$ was set using Movielens. The relevance threshold (γ) proposed in this paper (see Section 2.1) was set to 4 in the databases rated from 1 to 5, and it was set to 6 in the FilmAffinity experiments. Finally, the experiments were carried out using both users' and items' cross-validation.

A detailed explanation of the precision and recall quality measures applied to RS can be found in [12]. Let Z_u be the set of N items recommended to the user u, and $r_{u,i}$ the vote of user u to item i; assuming that all users accept N test recommendations:

$$precision_u = \frac{\#\{i \in Z_u | r_{u,i} \geqslant \theta\}}{N} \tag{11}$$

$$Recall_u = \frac{\#\{i \in Z_u | r_{u,i} \geqslant \theta\}}{\#\{i \in Z_u | r_{u,i} \geqslant \theta\} + \#\{i \in Z_u^c | r_{u,i} \neq \bullet \wedge r_{u,i} \geqslant \theta\}} \tag{12}$$

Fig. 8 shows the quality of the resulting recommendations. Graphs a, b, and c show the results of using MovieLens, Netflix, and FilmAffinity, respectively. As expected from the preliminary results and the actual design of the metric, *HwSimilarity* (labeled HW) provides a lower recommendation quality than JMSD but provides a significantly greater recommendation quality than the Pearson correlation.

As the number of recommendations N increases, the precisions of the metrics become equal, as expected. The incorporation of relevant items among the recommendations is more probable with an

[1] Hosted by the authors' research group. Open access at http://IntelligentSystems.eui.upm.es.

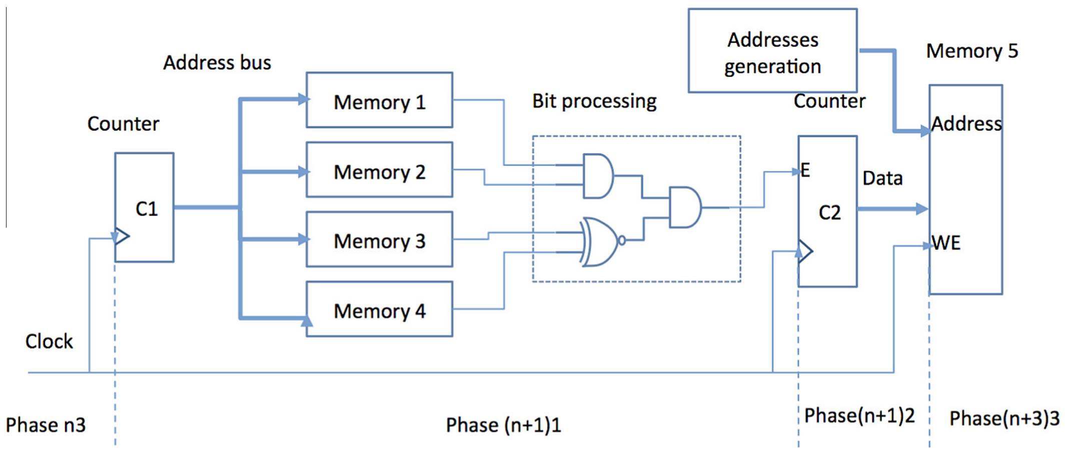

Fig. 6. Sequential hardware implementation of the *HwSimilarity* metric between 2 users.

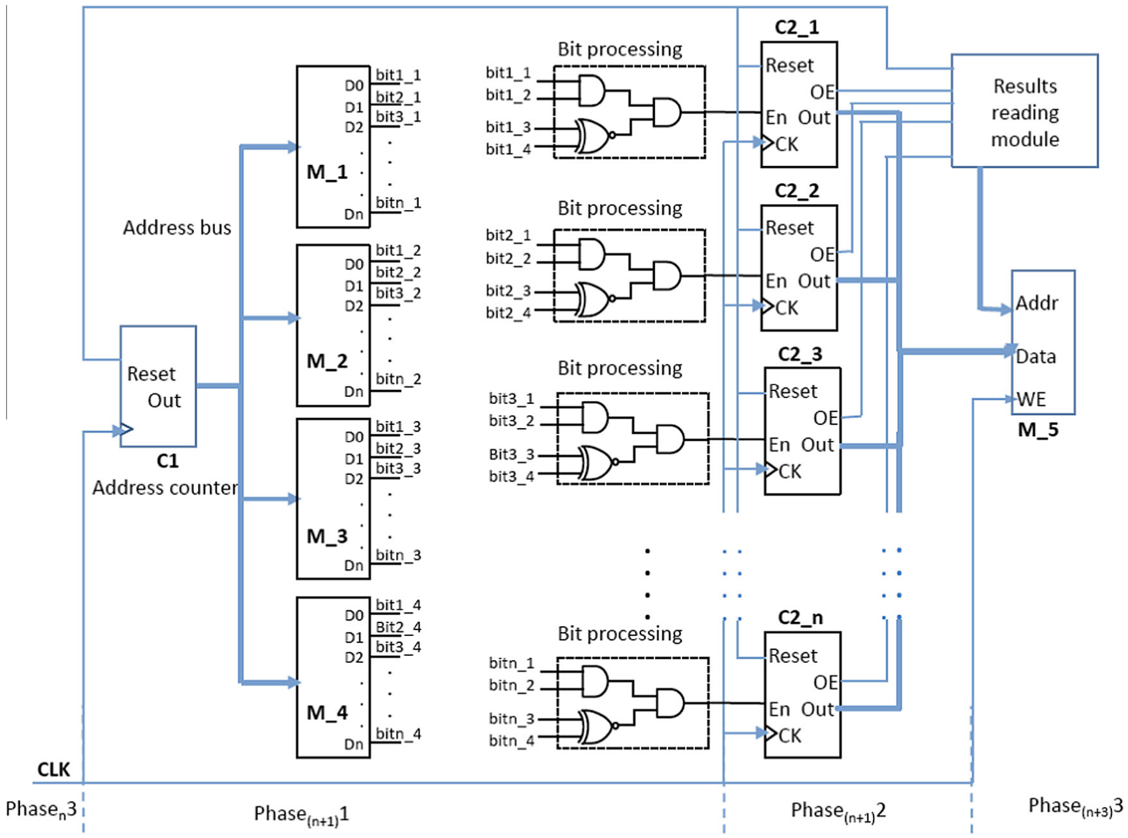

Fig. 7. Parallel hardware implementation of the *HwSimilarity* metric between an active user and the remaining users of the RS database.

Table 1
Main parameters used in the experiments.

	Precision/recall			Common parameters		
	N	θ	K	γ	Test users (%)	Test items (%)
Movielens 1 M	{2,...,20}	5	300	4	20	20
Netflix		5	350	4	10	20
FilmAffinity		9	350	6	10	20

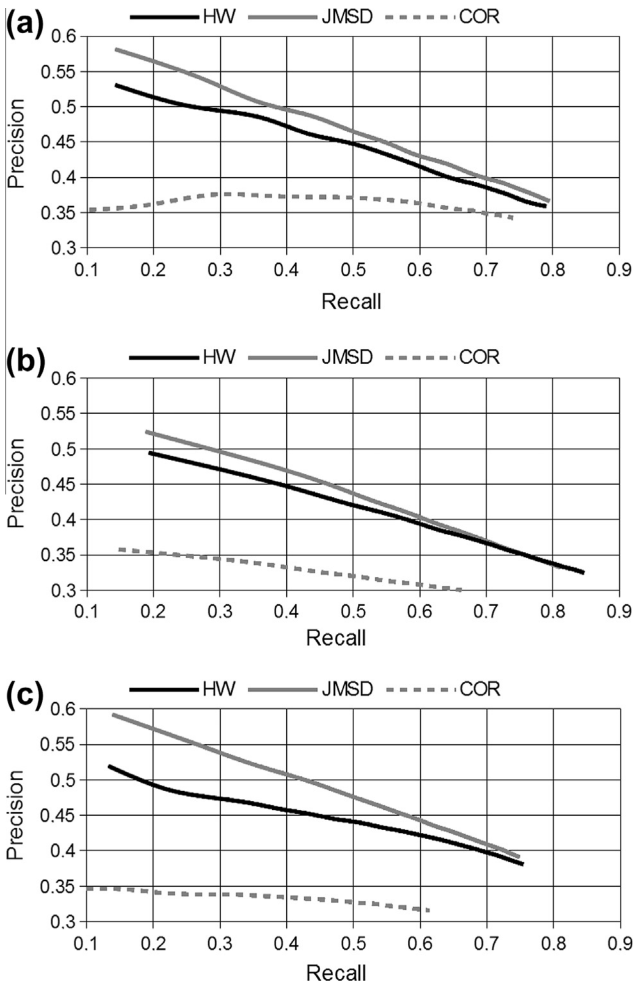

Fig. 8. Precision and recall results using (a) MovieLens, (b) Netflix, and (c) FilmAffinity. $N \in \{2 \cdots 20\}$.

increasing number of recommended items, i.e., a representative subset of relevant items.

The greatest difference between the recommendation qualities obtained by JMSD and *HwSimilarity* is attributable to the use of FilmAffinity (graph c), which suggests that the process of discretization from votes to relevant/irrelevant values is more distinct in FilmAffinity (ratings from 1 to 10) than in MovieLens and Netflix (ratings from 1 to 5).

The obtained percentages of quality loss from JMSD to *HwSimilarity* are 3.87%, 5.66% and 11.11% (Netflix, Movielens and FilmAffinity). These results show that the proposed similarity measure is appropriated when used on commercial RS databases, if ratings have a range not higher than 1–5.

4.2. Performance (time consuming) results

First, we calculate the improvement that is obtained during execution by utilizing the proposed metric. For this purpose, we process the *k*NN algorithm over the entire MovieLens database with software. We take the Pearson correlation and JMSD as baselines. Table 2 shows the processing times (absolute and relative to *HwSimilarity*) required for each of the similarity measures compared. The times indicated correspond to the number of seconds needed to calculate the similarities of all the test users with all the training users.

Table 2
Required processing time for each similarity measure (*HwSimilarity* software implementation, MovieLens database).

	COR	JMSD	HwSimil.
Absolute processing time (in seconds)	1.732	2.084	0.830
Relative processing time	+2.09	+2.51	1

Table 3
The amount of time required to process one item using *HwSimilarity*. "*n*" is the number of bit processing units in the parallel hardware approach.

Time (ns.)	Processor	Operating system
$7.2/n$	Parallel hardware	
7.2	Sequential hardware	
17.5	Intel Core 2 Quad Q6600, 2.4 GHz	Windows 7
19.7	Intel Core i7-3770, 3.4 GHz	Ubuntu
55.3	Intel Core 2 Duo P8800, 2.66 GHz	Mac OS X

By analyzing Table 2, we can determine that the metric designed (*HwSimilarities*) complies with the target performance restrictions required.

To assess the performance of the hardware approach, we calculate the time required to process only one item via Eq. (10) for various operating systems and processors. Table 3 lists the corresponding results.

Results in Table 2 show that the proposed metric (*HwSimilarity*) runs twice as fast as JMSD when computer processors (software) are used. Additionally to this speedup, Table 3 shows that sequential hardware runs more than twice as fast as the fastest analyzed computer. Moreover, the parallel hardware approach can reduce linearly execution time with the number of bit processing units (n) used.

5. Conclusions

Recommender systems that are based on the *k*NN collaborative filtering algorithm are common to commercial and academic domains. The latent problems of recommender systems include the following issues: the algorithm is not scalable, its execution requires considerable processing time, the size of the databases continually expands, and the recommendation requests from users do not conform to a uniform distribution. As an alternative to the *k*NN algorithm, the model-based collaborative filtering improves response time; however, it has the disadvantage of continuously downgrading the models.

With the goal of accelerating the execution of the *k*NN algorithm, this paper proposes a new similarity metric: *HwSimilarity*. This metric yields a recommendation quality that is extremely similar to those of the best existing metrics, and it can be processed using low-cost hardware circuits (as little as \$1). The proposed metric is approximately twice as fast as traditional metrics, such as the Pearson correlation. In addition to this intrinsic advantage, it allows hardware implementations that can reduce execution time by half (approximately, in the sequential version) and by a maximum of 2n times (parallel implementation, with n being the number of processing bit units used).

This paper provides designs for low-cost hardware implementations; these designs implement the *HwSimilarity* metric (sequential and parallel). For future studies, we suggest that the following issues be explored: (a) the creation of hardware devices (coarse-grained parallelization) that will help resolve simultaneous recommendation requests from users of one or more recommender systems and (b) the development of a prototype recommender system in which users recognize, in real time as they add new ratings, changes in the recommendations that they receive (recommender systems push technology).

Acknowledgements

This work has been supported by the Ministerio de Economía y Competitividad with the Project TIN2012-32682.

Our acknowledgement to the *Grouplens Research Group* and to the *FilmAffinity.com* and *Netflix* companies. The authors thankfully acknowledges the computer resources, technical expertise and assistance provided by the *Supercomputation and Visualization Center of Madrid (CesViMa)* and to the *Spanish supercomputation network*.

References

[1] G. Adomavicius, A. Tuzhilin, 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) (2005) 734–749.
[2] H.J. Ahn, A new similarity measure for collaborative filtering to alleviate the new user cold-starting problem, Information Sciences 178 (1) (2008) 37–51.
[3] D. Anand, K.K. Bharadwaj, Utilizing various sparsity measures for enhancing accuracy of collaborative recommender systems based on local and global similarities, Expert Systems with Applications 38 (5) (2011) 5101–5109.
[4] N. Antonopoulus, J. Salter, Cinema screen recommender agent: combining collaborative and content-based filtering, IEEE Intelligent Systems 2 (2006) 35–41.
[5] R.J. Barrientos, J.I. Gómez, C. Tenllado, M. Prieto, Heap based k-nearest neighbor search on GPUs, XXI Jornadas de Paralelismo (2010) 559–566.
[6] R.J. Barrientos, J.I. Gómez, C. Tenllado, M. Prieto, M. Marin, KNN query processing in metric spaces using GPUs, in: 17th International Conference on Parallel Processing, vol. 1, 2011, pp. 380–392.
[7] G. Beliakov, G. Li, Improving the speed and stability of the k-nearest neighbors method, Pattern Recognition Letters 33 (2012) 1296–1301.
[8] J. Bobadilla, F. Serradilla, A. Hernando, Collaborative filtering adapted to recommender systems of e-learning, Knowledge Based Systems 22 (2009) 261–265.
[9] J. Bobadilla, F. Serradilla, J. Bernal, A new collaborative filtering metric that improves the behavior of recommender systems, Knowledge-Based Systems 23 (2010) 520–528.
[10] J. Bobadilla, F. Ortega, A. Hernando, A collaborative filtering similarity measure based on singularities, Information Processing and Management 48 (2) (2012) 204–217.
[11] J. Bobadilla, F. Ortega, A. Hernando, J. Alcalá, Improving collaborative filtering recommender system results and performance using genetic algorithms, Knowledge-Based Systems 24 (8) (2011) 1310–1316.
[12] J. Bobadilla, A. Hernando, F. Ortega, J. Bernal, A framework for collaborative filtering recommender systems, Expert Systems with Applications 38 (12) (2011) 14609–14623.
[13] J. Bobadilla, A. Hernando, F. Ortega, A. Gutiérrez, Collaborative filtering based on significances, Information Sciences 185 (1) (2012) 1–17.
[14] J.S. Breese, D. Heckerman, C. Kadie, Empirical analysis of predictive algorithms for collaborative filtering, in: 14th Conf. on Uncertain. in Artif. Intell., Morgan Kaufmann, 1998, pp. 43–52.
[15] S.H. Cha, S.N. Srihari, A fast nearest neighbor search algorithm by filtration, Pattern Recognition 35 (2002) 515–525.
[16] J.L. Herlocker, J.A. Konstan, J.T. Riedl, L.G. Terveen, Evaluating collaborative filtering recommender systems, ACM Transactions on Information Systems 22 (1) (2004) 5–53.
[17] F. Hernández, E. Gaudioso, Evaluation of recommender systems: a new approach, Expert Systems with Applications 35 (3) (2008) 790–804.
[18] M.Z. Jahromi, E. Parvinnia, R. John, A method of learning weighted similarity function to improve the performance of nearest neighbor, Information Sciences 179 (2009) 2964–2973.
[19] H. Langseth, T.D. Nielsen, A latent model for collaborative filtering, International Journal of Approximate Reasoning 53 (4) (2012) 447–466.
[20] S.H. Li, B. Myaeng, M. Kim, A probabilistic music recommender considering user opinions and audio features, Information Processing and Management 43 (2) (2007) 473–487.
[21] M. Muja, D. Lowe, Fast approximate nearest neighbors with automatic algorithm configuration, in: International Conference on Computer Vision Theory and Applications, 2009, pp. 331–340.
[22] F. Ortega, J.L. Sánchez, J. Bobadilla, A. Gutiérrez, Improving collaborative filtering based recommender systems using Pareto dominance, Information Sciences 239 (2013) 50–61.
[23] C. Porcel, E. Herrera-Viedma, Dealing with incomplete information in a fuzzy linguistic recommender system to disseminate information in university digital libraries, Knowledge-Based Systems 23 (1) (2010) 32–39.
[24] C. Porcel, A. Tejeda-Lorente, M.A. Martínez, E. Herrera-Viedma, A hybrid recommender system for the selective dissemination of research resources in a Technology Transfer Office, Information Sciences, 2011, doi: http://dx.doi.org/10.1016/j.ins.2011.08.026 (in press).
[25] C. Silpa-Anan, R. Hartley, Optimized kd-trees for fast image descriptor matching, in: IEEE Conference on Computer Vision and Pattern Recognition, 2008, pp. 1–8.
[26] J. Serrano, E.H. Viedma, J.A. Olivas, A. Cerezo, F.P. Romero, A Google wave-based fuzzy recommender system to disseminate information in University Digital Libraries 2.0, Information Sciences 181 (8) (2011) 1503–1516.