A recommendation approach for programming online judges supported by data preprocessing techniques

Raciel Yera[1] · Luis Martínez[2]

Abstract The use of programming online judges (POJ) to support students acquiring programming skills is common nowadays because this type of software contains a large collection of programming exercises to be solved by students. A POJ not only provides exercises but also automates the code compilation and its evaluation process. A common problem that students face when using POJ is information overload, as choosing the right problem to solve can be quite frustrating due to the large number of problems offered. The integration of current POJs into e-learning systems such as Intelligent Tutoring Systems (ITSs) is hard because of the lack of necessary information in ITSs. Hence, the aim of this paper is to support students with the information overload problem by using a collaborative filtering recommendation approach that filters out programming problems suitable for students' programming skills. It uses an enriched user-problem matrix that implies a better student role representation, facilitating the computation of closer neighborhoods and hence a more accurate recommendation. Additionally a novel data preprocessing step that manages anomalous users' behaviors that could affect the recommendation generation is also integrated in the recommendation process. A case study is carried out on a POJ real dataset showing that the proposal outperforms other previous approaches.

✉ Luis Martínez
martin@ujaen.es

Raciel Yera
yeratoledo@gmail.com

1 University of Ciego de Ávila, Ciego de Ávila, Cuba

2 Computer Science Department, University of Jaén, Jaén, Spain

Keywords Programming online judges · Recommender systems · Collaborative filtering · Users' inconsistencies

1 Introduction

Programming online judges (POJs) are software tools that contain a large collection of programming exercises to be solved by their users (usually students), whose main purpose is to automate the compilation and evaluation processes for the users' solutions to the proposed problems. Over the last few years the popularity of POJs has increased, being successfully used in two different scenarios: training for students that participate in ACM-ICPC-like programming contests, and the systematic practice of programming skills for students in computer science colleges [27, 30, 41, 54].

Basically, traditional POJs present a sequential list of programming problems that users, at their own personalized pace, can choose and try to solve. In the last few years, world-wide acceptance of these tools has implied an increase in the amount of information associated with these kind of applications. As examples, among the more widely recognized POJ are the *Peking University Online Judge* (http://poj.org/), at the moment with 600000+ users and 3000+ problems, the *University of Valladolid Online Judge*, with 210000+ users and 1700+ problems (https://uva.onlinejudge.org), and the *Caribbean Online Judge* (http://coj.uci.cu/), with 23000+ users and 3400+ problems. Other relevant POJs are also the *SPOJ Online Judge* (http://www.spoj.com/), the *Timus Online Judge* (http://acm.timus.ru/), and the *Saratov State University Online Judge* (http://acm.sgu.ru/).

In these applications it is considered that the users with better performance are those that solve the most problems, because it subsequently implies a strong degree of achievement and satisfaction [45]. Such growing satisfaction would increase the learners' efforts and performance in problem solutions and programming subjects, according to a basic pedagogical rule ("learners' effort will increase if they are more satisfied") [37]. In this way, the users' learning process in computer science subjects is enhanced, as has been also pointed out in related works such as [45].

It is easy to see that the learning process in current POJs, based on the previous view *the more problems solved the better skills acquired* by the students, are far from other e-learning systems such as ITSs or ontology-supported e-learning approaches.

Nevertheless, nowadays the increasing popularity of POJs has caused an information overload regarding the amount of available problems. For this reason, usually many novice users fail to solve problems because they are not able to choose appropriate problems to solve, in accordance with their existing level of knowledge. This produces widespread discouragement in POJ participation, resulting in students losing any benefits that this application might have provided them.

A typical solution for information overload problems is the use of recommender systems in the corresponding scenario [1, 32]. Recommender systems (RS) are applications focused on providing users with the information items that best fit their preferences and needs in an overloaded search scenario. With this aim in mind, RS have been used in several scenarios such as e-commerce [23], e-learning [25], tourism [39], libraries [10], social networks [11], and other applications areas.

Specifically, e-learning recommender systems have become a widely-developed area in the last few years [16]. However, recent reviews have shown that most work in this area is focused on specific domains [12, 25]. These issues make the use of such previous developments to build a recommendation approach for suggesting problems in the current POJ scenario to overcome the information overload really difficult.

Moreover, it is also difficult to adapt typical approaches for building personalized e-learning systems, such as the already-mentioned cases of ITSs and ontology-supported e-learning approaches, to be used in POJs. ITSs [17, 44, 50], are dynamic and adaptive systems for personalized instruction based on the students' characteristics and behavior that involve a combination of various fields such as artificial intelligence, cognitive psychology and educational research. However, they tend to be composed of complex architectures that include a domain model, a student model, a teaching model, and interfaces [36]; therefore their modelling is not possible in current POJs because they do not manage the required information to build them (such as problem difficulties, user learning objectives, etc.).

On the other hand, ontologies are partial, explicit specifications of a conceptualization, representing by means of a declarative formalism, the knowledge of a domain in terms of classes, properties, axioms and instances of classes [19, 20]. Specifically, over the last few years they have been increasingly used to support e-learning systems [18] (e.g. recently Miranda et al. [35] pointed out that subject ontologies could be used to semantically organize typical items related to e-learning systems, such as learning objects, learning activities, learning goals, assessment events, and learning events, resulting in the improvement of the recommendation processes). However, once again current POJs do not contain such items, and therefore it is hard to combine them with ontological approaches, taking this task out of the scope of the current research paper.

Additionally, the lack of information previously indicated regarding learning objectives, difficulty of problems, and so on, makes the use of POJs in recommendation approaches such as content-based recommendation difficult [13, 31]. For these reasons, previous work focused on recommending problems to solve in POJs is based on collaborative filtering (CF) and does not depend on additional information [57, 61]. Therefore, the current research proposal is focused on improving the performance of this type of recommendation regarding previous proposals in the specialized literature, using a memory-based CF approach.

Memory-based collaborative filtering was pioneer in collaborative filtering research, and it is still widely-used due to its simplicity and performance [25]. This paper presents a memory-based collaborative filtering recommendation approach to support students trying to overcome information overload in POJs by personalizing the problems that they should solve according to their current skills, avoiding failures and frustrating experiences. Specifically, the novelty of this proposal is twofold:

1. This approach is based on a user-problem matrix that incorporates information related to the students' failures when they try to solve problems. The use of this enriched matrix implies a better student's profiling, facilitating the computation of closer neighborhoods and hence a more accurate recommendation.
2. It incorporates a data preprocessing step that detects and corrects anomalous users' behaviors that could affect the recommendation generation. This step is based on

the concept of *natural noise management* associated with users' preferences in recommender systems [29, 58–60].

Even though our proposal has been conceived and developed specifically for POJs, its novel ideas could be also applied to other e-learning recommendation domains associated with problem solving.

Finally the proposal is validated by a case study on a real dataset of a POJ in order to show its performance as compared with previous approaches.

The rest of the paper is structured as follows. Section 2 revises the required background for the proposal presentation, related to programming online judges, collaborative filtering, e-learning recommender systems, and natural noise management in recommender systems. Section 3 presents the recommendation approach for POJ, which is composed of three steps: extended user-problem matrix building (Section 3.1), extended user-problem matrix preprocessing (Section 3.2), and recommendation of problems (Section 3.3). Section 4 shows an experimental evaluation of the proposal, by using a dataset extracted from a POJ and comparing its results with other approaches. Finally, Section 5 concludes the contribution.

2 Background

This section presents the required background for this research, specifically it refers to programming online judges (Section 2.1), collaborative filtering (Section 2.2), e-learning recommender systems (Section 2.3), and natural noise management in recommender systems (Section 2.4).

2.1 Programming online judges

Programming online judges (POJs) are web applications that store a list of programming problems to be solved. The typical interaction between the user and the POJ is as follows:

1. The user selects a problem to be solved in the POJ.
2. The user writes a program as a solution for the selected problem.
3. The user uploads this solution to the POJ.
4. The POJ evaluates this solution by using predefined input-output data. If the solution is correct, the problem is considered as solved. Otherwise, the user can go to step 2 to try an alternative solution for the same problem, or go to step 1 to choose a different exercise to solve.

Recently, there have been several developments focused on the use of these applications to support programming subjects. Kurnia et al. [27] formally presents POJs as tools for the evaluation of source codes developed by students as problem resolution. Leal and Silva [28] present Mooshak, a system to support programming contest management which could also be used as an online judge. Verdú et al. [52] present EduJudge, which aims at developing the UVa Online Judge into an effective educational environment that integrates Moodle and the competitive e-learning tool QUESTOURnament. Similarly, Petit et al. [41] present Jutge.org, a new application composed of a large collection of exercises focused on teaching and learning programming subjects, for which popularity is still increasing. Llana et al. [30] present FLOP, a user-centered system for automatic exercise assessment which prioritizes the administration facilities, and as such outperforming previous proposals. Finally, Wang et al. [54] present OJPOT, a novel teaching idea that integrates online judges with practice oriented programming teaching, focusing on cultivating students' practical abilities through several kinds of programming practices. Beyond these works there are other important contributions that, although generally do not manage the term online judge, are focused on the automatic assessment of programming problems solutions. Some reviews in this area were presented by Ala-Mutka [2] and Caiza and Del Amo [7].

Our proposal aims at empowering programming online judges research and development by providing a tailored recommendation approach, supported by data preprocessing techniques.

2.2 Collaborative filtering

CF emerged at mid 90s as an effective paradigm for building recommender systems. CF methods are currently among the most widespread recommendation approaches [46], and in addition, as an added value, they receive less information as input than other paradigms like content-based [33] and knowledge-based [34].

The CF approaches have been divided in two main families: the memory-based CF approaches and the model-based CF approaches [1]. In the memory-based approaches [1], the recommendations are generated using preference information associated with users that are similar to the active user. On the other hand, model-based approaches [1] focused on discovering intermediate knowledge (such as association rules, graphs, and other forms of knowledge representation), are to be able to build a predictive model that is then used to generate final recommendations.

Even though many memory-based CF approaches have been developed over the last 20 years, it is still an active research area in recommender systems [32] and often applied in real-world recommender systems because of their simplicity, justifiability, efficiency, and stability [38]. Due to these desirable features, the recommendation approach presented in this paper is a memory-based approach.

2.3 Recommender systems in e-learning

E-learning scenarios have always been one of the main application areas for recommender systems. Therefore, several surveys have been published to condense the relevant research in this area [12, 25]. Regarding this large amount of information, this section will review the work related with the current context, which is the recommendation to support programming subjects.

As a result, De Oliveira et al. [14] presented a recommender system to support the programming practices, through recommending "classes" of activities. This task is treated as a multi-label classification approach, where the student's profile is associated with some of the mentioned "classes". "Math operators", "data types", "if structure", "while structure", and others could be cited as examples of these classes.

Hsiao et al. [22] presented JavaGuide as a system to guide students through the appropriate questions in a Java programming course. In the system, each question is composed of a small source code fragment, where the student has to introduce the final value for some variable, or determine the final output of the code. In addition, the authors investigated the real-time effect of the system in a group of people.

Klasnja-Milícevíc et al. [24] and Vesin et al. [53] described the recommendation module associated with an intelligent programming tutoring system, that could be automatically adapted to the students' knowledge levels. The system is able to recognize the students' learning styles, and with this aim, uses several domain ontologies. Its main purpose is to find access sequences for the activities, in order to use traditional CF techniques to recommend relevant links and actions during the learning process.

Ruiz-Iniesta et al. [47] presented a knowledge-based strategy to recommend educational resources in a programming course, in order to provide personalized access. The proposal depends on a resource description based on metadata standards, enriched by semantic information represented by ontologies, and also contextual user information.

In short, these proposals were conceived for specific scenarios and are high content-dependent, beyond the basic user-item interaction. In this way, in the current context associated with POJs, only the users' log is available when they try to solve the existing problems. Therefore, it is very difficult to adapt most of the previous research to this context.

However, the closest antecedents related to the current contribution are the proposals presented in [57, 61]. At first, the proposal introduced in [57] presents a user-based collaborative filtering approach to suggest problems to solve in a programming online judge. However, in contrast to our proposal, it is mainly focused on a direct application of the traditional collaborative filtering techniques in the mentioned context, and does not consider any kind of data preprocessing method. On the other hand, the work presented in [61] suggests the application of an item-based collaborative filtering approach to suggest problems to solve in this context. With this aim, the authors proposed the use of a binary user-problem matrix [61]. In the experimental section a direct comparison will be performed with these works, in order to show the advantages introduced by the current proposal.

2.4 Natural noise management in recommender systems

The natural noise management (NNM) [40, 59] is a relatively new research area in the RS research field. In contrast to the traditional research in RS, which aims at proposing new algorithms that directly improve the recommendation accuracy, the NNM approaches work toward this improvement by processing inconsistent user preferences. Therefore, they assume that the removal of noisy preferences implies a performance enhancement of recommendation algorithms.

Natural noise in recommender systems refers to those imprecise ratings unintentionally introduced by users, which do not reflect their real preferences, and that affect the recommendation result. Several authors have suggested that these ratings are introduced due to diverse causes such as personal conditions, social influences, the context, emotional state, or certain rating scales [4, 48].

Yera Toledo et al. [59] grouped the natural noise management approaches in two groups: 1) the methods that use additional information beyond the user-item matrix for the recommendation generation [5, 42], and 2) the methods that only use the user-item matrix with this aim in mind [9, 29, 58–60].

The referred research preprocess users' inconsistencies in recommender systems using the traditional recommendation scenario as a base [1]. However, several contributions have referred to the presence of inconsistencies associated with recommendation scenarios beyond the traditional rating matrix [26, 43]. In this way, the current research introduces a data preprocessing step to identify and correct anomalous user behaviors in the POJs scenario (which could be recognized as natural noise), in order to improve the recommendation accuracy in this context.

3 A recommendation approach for programming online judges

This section presents a novel recommendation approach for programming online judges (POJ), supported by the users' behavior when trying to solve the presented problems.

Specifically, the proposal receives as input a set of triples $< u, p, j >$, each one representing a user u attempt to solve a problem p, receiving as verdict the judgment j (which could be "accepted" if the problem was successfully solved, or "not accepted" in any other case). In order to facilitate the proposal presentation, the relation $D : (u, p) \rightarrow (n, solved)$ will be assumed, so that a corresponding user u and problem p is received as input, and is returned the number of times n that the user tries to solve the problem, and if the problem was finally solved by the user (solved).

Because of the lack of additional information in the current context, this approach is built using the memory-based CF recommendation paradigm. In addition to the advantages associated with this paradigm, presented in Section 2.2, it has also been widely used in e-learning scenarios [55, 56].

Figure 1 shows the general overview of the approach, which receives as input the relation D and the active user x, and returns as output the list of recommended problems to be solved by x. The following sections describe this approach in further detail. It is composed of three main steps: extended user-problem matrix building (Section 3.1), extended user-problem matrix preprocessing for natural noise management (Section 3.2), and problems recommendation (Section 3.3).

3.1 Extended user-problem matrix building

Firstly, the transformation of all available data into an appropriate format is required in order to apply a memory-based collaborative filtering approach. Therefore, we use a boolean user-problem matrix M that is filled with one values (1) if the corresponding user solves the corresponding item, and if not is assigned the zero value (0). Equation (1) shows this initialization procedure.

$$M[u, p] = D(u, p).solved \tag{1}$$

Nevertheless, the dynamic of the user-problem interactions in POJs suggests that the information behind this basic matrix M is not enough to calculate a proper user similarity that takes into account user knowledge and abilities. In this way, instead of the analysis of a simple similarity regarding accepted or not accepted problems by users, we propose the enrichment of the matrix M by considering the amount of unsuccessful solution attempts $(D(u, p).n)$, in addition to the final problem's accepted or not accepted judgment. This enrichment would allow the construction of a new matrix M^* that guarantees a more precise similarity calculation and consequently better recommendation accuracy.

Therefore, we propose converting the matrix M (containing the "accepted or not accepted" user-problem interaction values), into a matrix M^* composed of five different interaction values.

At first, we suggest transforming the former "not accepted" category $M[u, p] = 0$ in M, into the following three categories in M^*:

- $M^*[u, p] = 0$, when the user u has never tried to solve the problem p
- $M^*[u, p] = 1$, when the user u has not solved the problem p, and has tried it just few times

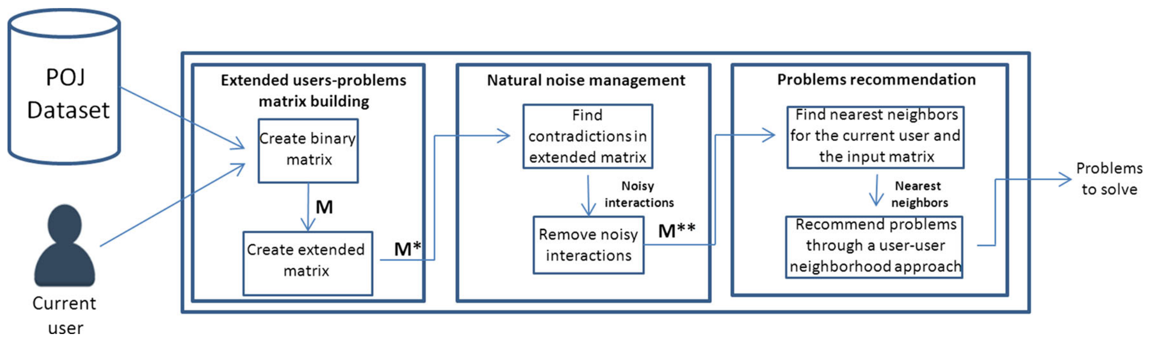
POJ Dataset
Current user
Extended users-problems matrix building
Create binary matrix
M
Create extended matrix
M*
Natural noise management
Find contradictions in extended matrix
Noisy interactions
Remove noisy interactions
M**
Problems recommendation
Find nearest neighbors for the current user and the input matrix
Nearest neighbors
Recommend problems through a user-user neighborhood approach
Problems to solve

Fig. 1 General overview of the recommendation approach for programming online judges

- $M^*[u, p] = 2$, when the user u has not solved the problem p, and has quite a few previous failed attempts

In addition, we also propose transforming the former "accepted" category $M[u, p] = 1$ in M, into the following two categories in M^*:

- $M^*[u, p] = 3$, when the user u solved the problem p, having failed many times beforehand
- $M^*[u, p] = 4$, when the user u solves the problem p quickly

$$M[u,p] = 0 \rightarrow M^*[u,p] = \begin{cases} \mathbf{0}, \text{ if } D(u,p).n = 0 \text{ (no solution attempts)} \\ \mathbf{1}, \text{ if } D(u,p).n \geq \lambda_{p\_1} \text{ (not solved, with many solution attempts)} \\ \mathbf{2}, \text{ if } D(u,p).n > 0 \text{ and } D(u,p).n < \lambda_{p\_1} \text{ (not solved, with few solution attempts)} \end{cases} \tag{2}$$

$$M[u,p] = 1 \rightarrow M^*[u,p] = \begin{cases} \mathbf{3}, \text{ if } D(u,p).n \geq \lambda_{p\_2} \text{ (solved with many solution attempts)} \\ \mathbf{4}, \text{ if } D(u,p).n < \lambda_{p\_2} \text{ (solved, with few solution attempts)} \end{cases} \tag{3}$$

Equations (2) and (3) formalize this transformation from matrix M to matrix M^*. These equations strongly depend on the thresholds λ_{p_1} and λ_{p_2}, which represent the boundary between the few and the many attempts categories in not solved and solved cases, respectively. In order to find proper initial values for these thresholds, we propose a problem-based strategy. In both cases we define the threshold as the average number of failures associated with all users that respectively have not solved or have solved the current problem p. Equations (6) and (7) illustrate the initialization strategies; NS_p and S_p being the set of users that have not solved/solved the problem p (4) and (5), and v any user in the dataset.

$$NS_p = \{v \mid D(v, p).solved = false\} \tag{4}$$

$$S_p = \{v \mid D(v, p).solved = true\} \tag{5}$$

$$\lambda_{p\_1} = \frac{\sum_{v \in NS_p} D(v, p).n}{|NS_p|} \tag{6}$$

$$\lambda_{p\_2} = \frac{\sum_{v \in S_p} D(v, p).n}{|S_p|} \tag{7}$$

3.2 Preprocessing natural noise in the extended user-problem matrix

This step aims at preprocessing inconsistent data associated with the user-problem matrix M^* to improve the recommendation generation, and specifically we will be focused on data unintentionally inserted by users, i.e. natural noise. Section 2.4 presented a brief survey on previous research related to natural noise management, concluding that this research is mainly focused on the general (e-commerce) recommendation scenario, and not on more specific contexts. However, there have been empirical studies showing that natural noise could affect and appear in any recommendation scenario [4].

Specifically in e-learning scenarios, Thai-Nghe et al. [51] and Klasnja-Milícevíc et al. [25] have pointed out that sometimes the students do not know how to solve the oriented activities, but guess them correctly; and in contrast, sometimes they know how to solve them, but make a mistake. These authors consider that this unexpected behavior should be taken into account in order to process any subsequent user data. Regarding this scenario and the causes related to the natural noise appearing, it can be concluded that these inconsistencies could be catalogued as natural noise.

Using these previous works as a basis in the POJ context, we characterize natural noise as those cases where the user does not solve a problem using the expected required effort according to: 1) his/her current knowledge levels, and 2) the difficulty of the current problem. These circumstances could be caused due to diverse factors, such as guessing, plagiarism, or the appearance of unexpected slip-ups in the application of well-acquired knowledge.

In order to identify and process this inconsistent behavior in the matrix M^*, we propose an approach that is basically presented in Fig. 2. It initially performs a classification for users, problems and interactions, and it then focuses on finding contradictions between these classifications. Afterwards, the detected contradictions are used to find inconsistent interactions (i.e. noisy interactions), which are finally corrected to obtain a new set of interactions with mitigated noise.

Hence, to classify the interactions we use the criteria presented in Section 3.1, specifically in (2) and (3). Therefore, the initial interactions' classification matches the categories associated with the matrix M^*, presented in the previous section.

On the other hand, it is necessary to propose a classification approach for the users and problems. We suggest identifying users according to the following tendencies:

Users' tendencies:

1. *Precise user*: the user tends to solve most of the finally solved problems quickly
2. *Imprecise user*: the user tends to have many failed solution attempts, prior to attaining the final problem solution
3. *Variable user*: the user's behavior varies between the two previous categories

Similarly, we suggest identifying problems according to the following tendencies:

Problems' tendencies:

1. *Easy problem*: a problem where users tend to need only a few failed solution attempts to finally solve it
2. *Difficult problem*: a problem where the users tend to need several previous failed solution attempts prior to attaining the final solution
3. *Variable problem*: a problem in which behavior varies between the two previous categories

To facilitate the formal definition for these tendencies in order to perform the user and problem classification, we propose grouping the problems solved by user u (those problems p associated with the categories $M^*[u, p] = 3$ and $M^*[u, p] = 4$). Specifically, we group them in two sets:

1. Set of problems solved by user u, needing too many attempts, $C1_u$: $C1_u = \{p \mid M^*[u, p] = 3\}$
2. Set of problems solved by user u, needing few attempts, $C2_u$: $C2_u = \{p \mid M^*[u, p] = 4\}$

In addition, we group the users that solved the problem p, into the following two sets:

1. Set of users that solved the problem p, needing too many attempts, $C1_p$: $C1_p = \{u \mid M^*[u, p] = 3\}$
2. Set of users that solved the problem p, needing few attempts, $C2_p$: $C2_p = \{u \mid M^*[u, p] = 4\}$

Table 1 formally defines the mentioned tendencies associated with users and problems, based on the cardinalities of these sets $C1_u$, $C2_u$, $C1_p$, and $C2_p$. Specifically, the *precise user* is defined as one whose amount of problems solved needing few attempts ($|C2_u|$), is equal or bigger than twice the amount of problems solved needing too many attempts ($|C1_u|$); and the *imprecise user* is defined as the reverse case. On the other hand, a problem is defined as *easy*, if the number of users that solved it with few attempts ($|C2_p|$) is bigger or equal to twice the amount of users that solved it with too many attempts ($|C1_p|$), and the *difficult problem* is defined as the reverse case. Finally, the *variable user* and *problem* for those cases in which any of the previous conditions are not respectively verified have been defined.

The initialization of the coefficients in Table 1 (also used later in (8) and (9)), is a hard task because the optimal values necessarily depend on the nature of the data associated with the POJ. Therefore, we focus on assigning suitable initial values supported by different experiments that we carried out and common sense as can be seen in the following points:

1. It is clear that the coefficient should be higher than 1, because a value equal to 1 would imply that there were not any users or items belonging to the variable categories. On the other hand, values lower than 1 could imply that some users or items were simultaneously classified in more than one class.

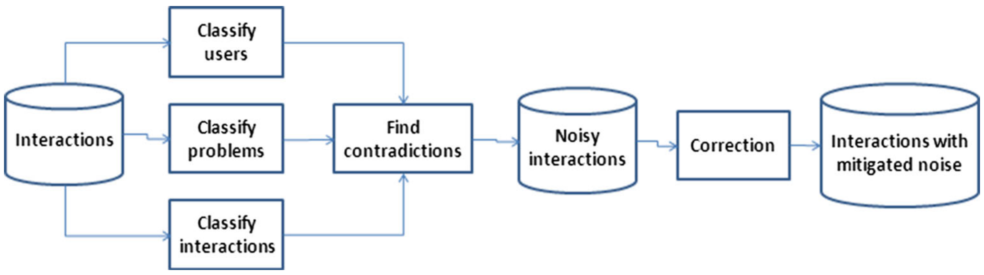

Fig. 2 The proposed approach for natural noise management in the presented scenario

2. In addition, high values for the coefficient would produce a very restrictive classification implying that most of the users and items were classified as variable, and therefore they were not checked for possible inconsistencies, as it will be shown below in (8) and (9)

Therefore, we choose the value 2 as a promising initial value for the coefficients. It is worthy to note that in addition to the previous reasoning, we have empirically explored the performance associated with different coefficient values through experimentation, usually obtaining the best performance for values equal or very close to the value 2.

Once each interaction, user, and problem have been classified, this information is used to find *noisy* interactions based on the definition of two groups of homologous classes (Table 2). In this way, the classes *precise* and *easy* are homologous because in both cases the interaction $M^*[u, p] = 4$ represents the majority of the interactions; and the classes *imprecise* and *difficult* are homologous because in this case $M^*[u, p] = 3$ represents the majority. We note that the literature related to the users' performance in programming online judges suggests the presence of these user and problem classes, and the presented association between them [6, 49].

Using these groups, we detect *noisy* interactions by assuming that for those cases where a user tries to solve a problem, and both the user and the problem belong to the same group of homologous classes, then the associated interaction category should also belong to this group. Therefore, a certain interaction where the corresponding user and problem have homologous classes, but the interaction category is not homologous with both classes, is then considered as natural noise.

Specifically, a *precise user* that solved an *easy problem*, should perform this task by needing few attempts ($M^*[u, p] = 4$), being considered noise if he/she performs it needing too many attempts ($M^*[u, p] = 3$). On the contrary, if an *imprecise user* solves a *difficult problem*, then it is expected that too many previous attempts were needed ($M^*[u, p] = 3$), being considered a noisy interaction if it was achieved requiring only a few attempts ($M^*[u, p] = 4$).

Afterwards, our natural noise management approach proposes the correction of the interaction categories detected as noisy, by assigning the homologous category according to the corresponding user and problem classes. Equations (8) and (9) summarize this correction strategy respectively applied to each group in Table 2, returning a matrix M^{**} of interactions with mitigated noise.

$$M^*[u,p] = 3 \rightarrow M^{**}[u,p] = \begin{cases} \mathbf{4}, \text{ if } |C2_u| \geq 2|C1_u| \text{ and } |C2_p| \geq 2|C1_p| \\ \mathbf{3}, \textit{in other cases} \end{cases} \tag{8}$$

$$M^*[u,p] = 4 \rightarrow M^{**}[u,p] = \begin{cases} \mathbf{3}, \text{ if } |C1_u| \geq 2|C2_u| \text{ and } |C1_p| \geq 2|C2_p| \\ \mathbf{4}, \textit{in other cases} \end{cases} \tag{9}$$

It is remarkable that the noise management approach was proposed by taking into account the two interaction categories associated with the successful problem solution, because they directly match the tendencies associated with the users or items, presented in Table 1. Although we do not discard the suitability of searching for noisy interaction in the "no solution" categories ($M^*[u, p] = 0$, $M^*[u, p] = 1$, and $M^*[u, p] = 2$, in (2)), the definition of homologous user and item classes for these categories were required, which is not an easy task. A deeper analysis of this will be left for future research.

3.3 Recommendation of Problems

This last step is supported by a traditional memory-based CF paradigm focused on the enriched matrix M^{**}. Being x the active user for providing recommendations, this step is composed of three phases:

1. Find the top k similar users regarding user x, according to M^{**}
2. For each problem not solved by x, calculate a score according to the top k similar users
3. Recommend the problems with the highest scores

The rest of this section presents these phases in further detail.

Find the top k similar users regarding user x, according to M^{**}: At first, it is necessary to represent the user profile for x, which is composed of the values of the interactions

Table 1 Classes for users and problems

User classes	
Precise	$\lvert C2_u\rvert \geq 2\lvert C1_u\rvert$
Imprecise	$\lvert C1_u\rvert \geq 2\lvert C2_u\rvert$
Variable	In other cases
Item classes	
Easy	$\lvert C2_p\rvert \geq 2\lvert C1_p\rvert$
Difficult	$\lvert C1_p\rvert \geq 2\lvert C2_p\rvert$
Variable	In other cases

Table 2 Homologous classes

	User classes	Item classes	Interaction category
Group 1	Precise	Easy	$M^*[u, p] = 4$
Group 2	Imprecise	Difficult	$M^*[u, p] = 3$

between user x and any problem p in the dataset, i.e. the row associated with user x in the matrix M^{**}.

To find the top k similar users regarding x, at first a similarity function must be chosen. To this effect, we select the Simple Matching Coefficient (SMC) [3], which is a function previously used to compare users in recommender systems, specifically in categorical scenarios like that currently presented. Specifically, we use a modification of the SMC function that discards the contribution of the category $M^{**}[u, p] = 0$ to the final similarity calculation. This modification is explained as follows.

S_{xu} being the set of problems where both the user x and the user u have the same interaction category, which in addition is different from the zero category ($M^{**}[u, p] = 0$) (10):

$$S_{xu} = \{p \mid M^{**}[u, p] = M^{**}[x, p] \text{ and } M^{**}[x, p] \neq 0\} \tag{10}$$

And D_{xu} the set of problems where at least the user x or the user u have some interaction in a category which is different from the zero category (11):

$$D_{xu} = \{p \mid M^{**}[u, p] \neq 0 \text{ or } M^{**}[x, p] \neq 0\} \tag{11}$$

The similarity between x and u is then defined according to (12):

$$sim(x, u) = \frac{S_{xu}}{D_{xu}} \tag{12}$$

Considering that the zero category ($M^{**}[u, p] = 0$) means *no solution attempts*, it is clear that the common presence of this category when two users are compared, does not necessarily indicate closeness. Therefore, this modification guarantees the similarity calculation only based on the remaining categories.

Once the similarity values between x and all users are calculated, the users with the top k higher values are selected as similar users and therefore employed in the following phases of the step.

For each problem not solved by x, a score is calculated according to the top k similar users: Once the list of top k similar users is obtained, these profiles are used to calculate a score w_p for each problem p not solved by user x. This score is calculated as the sum of the similarities between x and each neighbor that solves p (13). This phase uses the matrix M, which directly represents the success or failure in a problem solution.

$$w_p = \sum_{u \in top\_k(x)} sim(x, u) * M[u, p] \tag{13}$$

Recommend the problems with the highest scores: This final phase focuses on recommending the top n problems p with the highest scores w_p, to the active user x.

4 Experimental study

In order to evaluate the current proposal, offline experiments will be developed using a real dataset obtained from the Caribbean Online Judge from the period 2009-2010. This dataset is composed of 1910 users, 584 problems, and more than 148000 user attempts to solve problems. The average number of problems solved by each user is 30.24 problems, with a standard deviation of 56.39. These values suggest a high sparsity across all users in the number of solved problems, which is a typical feature associated with recommender system environments in relation to the interactions between users and items. Figure 3 shows a histogram reflecting the number of solved problems (y-axis) for each user (x-axis), clearly showing a long-tailed distribution that also characterizes data in common recommendation scenarios.

The experimental framework was developed according to the procedure suggested by Gunawardana and Shani [21]. In this case the training and test sets were created using the following procedure. For each user it randomly divides all the solved problems in two sets, adding the first one to the training set, and the second one to the test set. In

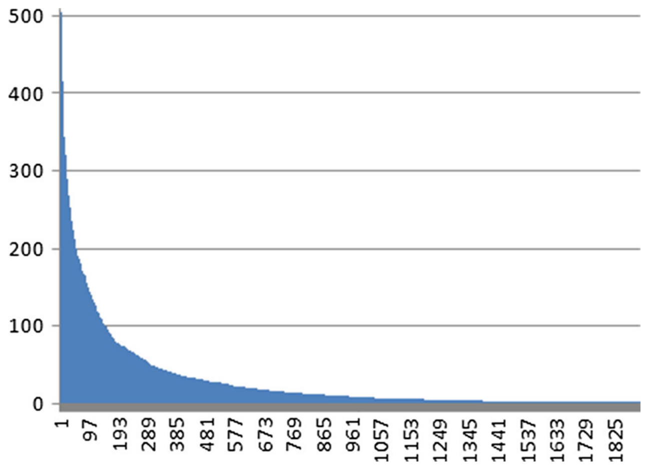

Fig. 3 Long-tailed histogram reflecting the number of solved problems (*y-axis*), for each user (*x-axis*). The users were sorted downwardly according to their amount of solved problems

Table 3 Possible recommendation results, regarding solved and recommended problems

	Recommended	**Not recommended**
Solved	True-Positive (tp)	False-Negative (fn)
Not solved	False-Positive (fp)	True-Negative (tn)

this way, both the training and the test set would contain data from all users. Hence, the experiments were developed by averaging the accuracy results of several training-test partitions created through this procedure, to avoid any undesirable bias in the distribution of the user data. Therefore, this procedure could be classified as a repeated user-driving hold-out approach [8]. Finally, the proposal also receives the information associated with the failed solutions for each user-problem pair as input, therefore this information is also stored independently.

Several evaluation measures have been proposed to evaluate the accuracy of the top-n recommendation task [21]. In this research we use the F1 measure, which is probably the most popular and generally-accepted [21]. F1 (14) is defined in terms of Precision and Recall measures (15) and (16), which in addition, as is presented in Table 3, depend on the amount of recommended problems that were solved (precision) and the amount of solved problems that were recommended (recall).

$$F1 = \frac{2 * precision * recall}{precision + recall} \tag{14}$$

$$precision = \frac{\#tp}{\#tp + \#fp} \tag{15}$$

$$recall = \frac{\#tp}{\#tp + \#fn} \tag{16}$$

In this work, for each user, given the list of ”solved” problems in the training set and the data regarding failures, the proposal generates a list of recommended problems. This list and the solved problems located in the test set are then used to calculate the current user's F1 value. At last, all the users' F1 values are averaged to obtain the final F1 value associated with the proposal. This process is repeated 10 times, obtaining 10 different F1 values that are averaged to obtain the final accuracy value to register.

Regarding the current proposal and previous research, the purposes of the evaluation are focused on the following main objectives:

- To evaluate the performance of the proposal across different sizes of the amount of neighbors used in the recommendation generation phase (k).

  Therefore, we evaluate the proposal varying the size of the used neighborhood (k) in the range [90, 170] with step 10. This evaluation was done for different sizes n of the recommendation list, modifying n specifically in the range [5, 40] with step 5. Figure 4 shows the results of this experiment, showing the sensitivity of k for different values of n in each graph. These results conclude that in most cases, the accuracy tends to increase when k increases in the range [90, 130], and become stable for $k > 130$. Therefore, in the remaining experiments we will set $k = 130$.
- To verify whether the matrix enrichment (Section 3.1) and the natural noise management process (Section 3.2), introduce an accuracy improvement in the recommendation generation.

  Consequently, we initially verify the difference between the neighborhoods used in the recommendation generation associated with each user x according to the original binary matrix M ($neighbors_x^M$), and the

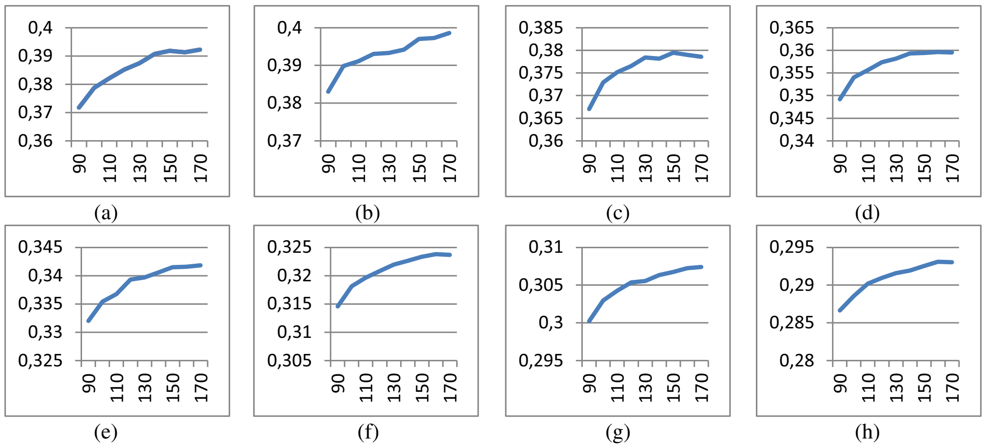

Fig. 4 Performance of the proposal, according to F1, for different sizes k of the used neighborhood (the x axis). **a** Top 5 recommendations. **b** Top 10 recommendations. **c** Top 15 recommendations. **d** Top 20 recommendations. **e** Top 25 recommendations. **f** Top 30 recommendations. **g** Top 35 recommendations. **h** Top 40 recommendations

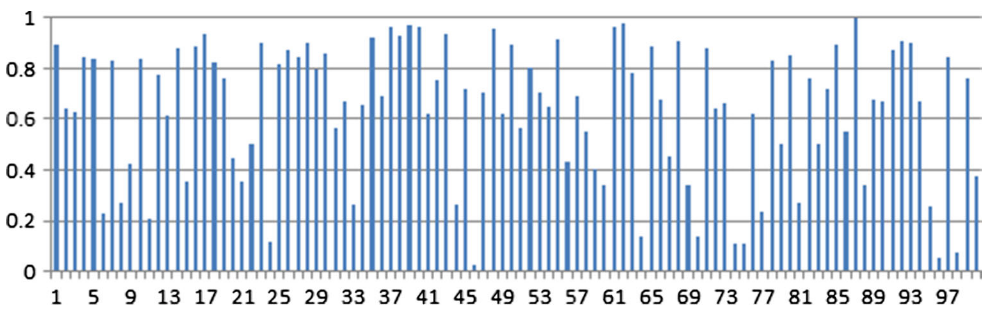

Fig. 5 Overlapping degree between the neighborhoods generated for a simple random sample of 100 users in the dataset, according the matrix M, and the matrix M^{**}

same neighborhoods but according to the matrix M^{**} ($neighbors_x^{M^{**}}$). To this purpose, we define the overlapping degree between both sets of neighborhoods for a user x (17), as the ratio between the cardinality of its intersection, and the initial size of the neighborhood (the k value, in this case $k = 130$). This ratio would indicate a low or a high effect of the proposal in Sections 3.1 and 3.2, in the formation of different and possibly more precise neighborhoods.

$$overlapping_x = \frac{|\,neighbors_x^{M} \cap neighbors_x^{M^{**}}\,|}{k} \tag{17}$$

Figure 5 shows this overlapping degree for a random sample of 100 users in the dataset (represented on the horizontal axis), which varies from values close to 0, to 1. Taking into account the 1910 users in the dataset, the average overlapping was 0.63 with a standard deviation of 0.22. Therefore, these results prove that the procedures proposed in Sections 3.1 and 3.2, definitively imply the formation of a different neighborhood in contrast to that associated with the original rating matrix M. However, it is necessary to verify whether these new neighborhoods introduce an accuracy improvement in the recommendation generation. With this aim in mind, we will compare our current proposal (Proposal, in Table 4) with a similar procedure that uses only the matrix M in the neighborhood formation step (Binary, in Table 4).

- To verify that the proposal outperforms related previous works, across different sizes of the top-n recommendation list.

  Eventually, we compare it to related works presented in [57] (UCF-OJ, in Table 4), and at [61] (ICF-OJ, in Table 4). The referred Table 4 shows these comparison results, varying the size of the list with top-n recommendations in the range [5, 40] with step 5. In all cases, our proposal notably outperforms the other approaches. Therefore, it proves that the dynamic matrix enrichment and the natural noise management processes are useful tools in order to outperform the recommendation generation in the POJ scenario.
- To evaluate the independent effect of the dynamic matrix enrichment and the natural noise management process

  We develop a new experiment that evaluates the proposal incorporating the matrix enrichment step (Section 3.1), but without the natural noise management (NNM) step (Section 3.2). Afterwards, we compare its result with the baseline (Binary) and the final proposal (Proposal), both previously evaluated. Table 5 presents the results of this comparison, showing that the application of the proposal without the NNM step (i.e. using only the matrix M^*) introduces important improvements in relation to the baseline (Binary, with matrix M). However, the improvement is not similarly clear at the comparison of the performance of the proposal without the NNM step (M^*), and incorporating the NNM step (final proposal, with matrix M^{**}), the final proposal being expected to have the best performance.

  To clarify this issue we have applied a statistical test, specifically the Wilcoxon non-parametric test, widely used in pairwise comparison [15, 21]. In order to obtain more representative samples, instead of using the data in Table 5, we repeat the executions by modifying the

Table 4 Comparison between the final proposal and related works, according to F1 values with $k = 130$

	5	10	15	20	25	30	35	40
Proposal	**0.3875**	**0.3933**	**0.3784**	**0.3582**	**0.3397**	**0.3220**	**0.3055**	**0.2916**
Binary	0.3614	0.3687	0.3540	0.3352	0.3189	0.3026	0.2895	0.2767
UCF-OJ	0.3833	0.3899	0.3736	0.3543	0.3367	0.3194	0.3035	0.2890
ICF-OJ	0.3602	0.3624	0.3494	0.3348	0.3191	0.3058	0.2932	0.2808

Table 5 Comparison between the baseline, the proposal without NNM, and the final proposal according to F1 values with $k = 130$

	5	10	15	20	25	30	35	40
Binary (M)	0.3614	0.3687	0.3540	0.3352	0.3189	0.3026	0.2895	0.2767
Proposal without NNM (M^*)	0.3866	**0.3936**	0.3773	0.3573	0.3393	0.3219	**0.3058**	0.2915
Proposal (M^{**})	**0.3875**	0.3933	**0.3784**	**0.3582**	**0.3397**	**0.3220**	0.3055	**0.2916**

size n of the recommendation list in the range [1, 40] with step 1, in contrast to the previous ones done in the range [5, 40] with step 5.

Once these executions were completed, the Wilcoxon test was applied to the results associated with the final proposal (obtained through M^{**}), and the proposal without NNM (obtained through M^*). This test shows as result that for 25 of the 40 comparisons, the final proposal obtains the best performance, being a statistically significant difference as $p < 0.05$. Therefore, this experiment verifies that both the dynamic matrix enrichment and the NNM approach play an important role in the proposal, because in both cases improvements are introduced in the recommendation performance.

5 Conclusions and future works

This paper presents a recommendation approach to suggest problems to solve in a POJ, supported by data preprocessing techniques. Specifically, the proposal is composed by three main steps: 1) the extended user-problem matrix building, 2) the extended user-problem matrix preprocessing for natural noise management, and 3) the problem recommendation. The experimental results show that steps 1) and 2) guarantee the formation of a more precise neighborhood, positively impacting the recommendation accuracy in relation to the related previous research. To the best of our knowledge this research is pioneer in the investigation field related to recommender systems in the POJ domain, and also related to the use of natural noise management techniques in an e-learning recommendation scenario.

Our further research will be focused on the following points: 1) proposing and evaluating more advanced similarity metrics that incorporate contextual information in POJs, such as time. 2) the use of fuzzy tools to introduce a better user and problem characterization in the natural noise management step, following the ideas presented in [60], and 3) the generalization of natural noise management process to other e-learning recommendation scenarios.

Acknowledgments This research work was partially supported by the Research Project TIN2015-66524-P.

References

1. Adomavicius G, Tuzhilin AT (2005) Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. IEEE Trans Knowl Data Eng 17(6):734–749
2. Ala-Mutka KM (2005) A survey of automated assessment approaches for programming assignments. Comput Sci Educ 15(2):83–102
3. Amatriain X, Pujol JM (2015) Data mining methods for recommender systems. In: Recommender Systems Handbook. Springer, pp 227–262
4. Amatriain X, Pujol JM, Oliver N (2009a) I like it... i like it not: Evaluating user ratings noise in recommender systems. In: User modeling, adaptation, and personalization. Springer, pp 247–258
5. Amatriain X, Pujol JM, Tintarev N (2009b) Rate it again: increasing recommendation accuracy by user re-rating. In: Proceedings of the third ACM conference on Recommender systems. ACM, pp 173–180
6. Arefin AS (2006) Art of Programming Contest. Gyankosh Prokashonia
7. Caiza J, Del Amo J (2013) Programming assignments automatic grading: Review of tools and implementations. In: Proceedings of INTED, vol 2013, pp 5691–5700
8. Campos PG, Díez F, Cantador I (2014) Time-aware recommender systems: a comprehensive survey and analysis of existing evaluation protocols. User Model User-Adap Inter 24(1-2):67–119
9. Castro J, Yera Toledo R, Martínez L (2016) An empirical study of natural noise management in group recommendation systems. Decision Support Systems. doi:10.1016/j.dss.2016.09.020
10. Chen LC, Kuo PJ, Liao IE (2015) Ontology-based library recommender system using mapreduce. Clust Comput 18(1):113–121
11. Christensen I, Schiaffino S (2014) Social influence in group recommender systems. Online Inf Rev 38(4):524–542
12. Dascalu MI, Bodea CN, Mihailescu MN, Tanase EA, Ordoñez de Pablos P (2016) Educational recommender systems and their application in lifelong learning. Behav Inform Technol 35(4):290–297
13. De Maio C, Fenza G, Gaeta M, Loia V, Orciuoli F, Senatore S (2012) Rss-based e-learning recommendations exploiting fuzzy fca for knowledge modeling. Appl Soft Comput 12(1):113–124
14. De Oliveira MG, Ciarelli PM, Oliveira E (2013) Recommendation of programming activities by multi-label classification for a formative assessment of students. Expert Syst Appl 40(16):6641–6651
15. Derrac J, García S, Molina D, Herrera F (2011) A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. Swarm Evol Comput 1(1):3–18
16. Drachsler H, Verbert K, Santos OC, Manouselis N (2015) Panorama of recommender systems to support learning. In: Recommender systems handbook. Springer, pp 421–451
17. Fenza G, Orciuoli F (2016) Building pedagogical models by formal concept analysis. In: International Conference on Intelligent Tutoring Systems. Springer, pp 144–153

18. Gaeta M, Orciuoli F, Paolozzi S, Salerno S (2011) Ontology extraction for knowledge reuse: The e-learning perspective. IEEE Trans Syst Man Cybern Part A Syst Humans 41(4):798–809
19. Gruber TR (1995) Toward principles for the design of ontologies used for knowledge sharing? Int J Hum Comput Stud 43(5):907–928
20. Guarino N, Giaretta P (1995) Ontologies and knowledge bases towards a terminological clarification. In: Towards very large knowledge bases: knowledge building & knowledge sharing. IOS Press, pp 25–32
21. Gunawardana A, Shani G (2009) A Survey of Accuracy Evaluation Metrics of Recommendation Tasks. J Mach Learn Res 10:2935–2962
22. Hsiao IH, Sosnovsky S, Brusilovsky P (2010) Guiding students to the right questions: adaptive navigation support in an e-learning system for java programming. J Comput Assist Learn 26(4):270–283
23. Huang Z, Zeng D, Chen H (2007) A comparison of collaborative-filtering recommendation algorithms for e-commerce. IEEE Intell Syst 5:68–78
24. Klaśnja-Milićević A, Vesin B, Ivanović M, Budimac Z (2011) E-learning personalization based on hybrid recommendation strategy and learning style identification. Comput Educ 56(3):885–899
25. Klaśnja-Milićević A, Ivanović M, Nanopoulos A (2015) Recommender systems in e-learning environments: a survey of the state-of-the-art and possible extensions. Artif Intell Rev 44(4):571–604
26. Krishnan S, Patel J, Franklin MJ, Goldberg K (2014) A methodology for learning, analyzing, and mitigating social influence bias in recommender systems. In: Proceedings of the 8th ACM Conference on Recommender systems. ACM, pp 137–144
27. Kurnia A, Lim A, Cheang B (2001) Online judge. Comput Educ 36(4):299–315
28. Leal JP, Silva F (2003) Mooshak: a web-based multi-site programming contest system. Software: Practice and Experience 33(6):567–581
29. Li B, Chen L, Zhu X, Zhang C (2013) Noisy but non-Malicious user detection in social recommender systems. World Wide Web 16(5-6):677–699
30. Llana L, Martin-Martin E, Pareja-Flores C, Velázquez-Iturbide JÁ (2014) Flop: a user-friendly system for automated program assessment. Journal of Universal Computer Science 20(9):1304–1326
31. Lops P, Gemmis M, Semeraro G (2011) Content-based recommender systems: State of the art and trends. In: Recommender systems handbook springer US, chap, vol 3, pp 73-105
32. Lu J, Wu D, Mao M, Wang W, Zhang G (2015) Recommender system application developments: a survey. Decis Support Syst 74:12–32
33. Martínez L, Pérez LG, Barranco M (2007) A multigranular linguistic content-based recommendation model. Int J Intell Syst 22(5):419–434
34. Martínez L, Barranco MJ, Pérez LG, Espinilla M (2008) A knowledge based recommender system with multigranular linguistic information. International Journal of Computational Intelligence Systems 1(3):225–236
35. Miranda S, Orciuoli F, Sampson DG (2016) A skos-based framework for subject ontologies to improve learning experiences. Comput Hum Behav 61:609–621
36. Murray T (1999) Authoring intelligent tutoring systems: an analysis of the state of the art. Int J Artif Intell Educ 10:98–129
37. Nadolski RJ, Van den Berg B, Berlanga AJ, Drachsler H, Hummel HG, Koper R, Sloep PB (2009) Simulating light-weight personalised recommender systems in learning networks: a case for pedagogy-oriented and rating-based hybrid recommendation strategies. Journal of Artificial Societies and Social Simulation 12(1):4
38. Ning X, Desrosiers C, Karypis G (2015) A comprehensive survey of neighborhood-based recommendation methods. In: Recommender Systems Handbook. Springer, pp 37–76
39. Noguera J, Barranco M, Segura R, Martínez L (2012) A mobile 3d-gis hybrid recommender system for tourism. Inf Sci 215:37–52
40. O'Mahony MP, Hurley NJ, Silvestre G (2006) Detecting noise in recommender system databases. In: Proceedings of the 11th international conference on Intelligent user interfaces. ACM, pp 109–115
41. Petit J, Giménez O, Roura S (2012) Jutge. org: an educational programming judge. In: Proceedings of the 43rd ACM technical symposium on Computer Science Education. ACM, pp 445–450
42. Pham HX, Jung JJ (2013) Preference-based user rating correction process for interactive recommendation systems. Multimedia tools and applications 65(1):119–132
43. Piramuthu S, Kapoor G, Zhou W, Mauw S (2012) Input online review data and related bias in recommender systems. Decis Support Syst 53(3):418–424
44. Polson MC, Richardson JJ (2013) Foundations of intelligent tutoring systems. Psychology Press
45. Regueras LM, Verdú E, Muñoz MF, Pérez MA, De Castro JP, Verdú MJ (2009) Effects of competitive e-learning tools on higher education students: a case study. IEEE Trans Educ 52(2):279–285
46. Ricci F (2015) Recommender systems handbook. Springer, Shapira B
47. Ruiz-Iniesta A, Jimenez-Diaz G, Gomez-Albarran M (2014) A semantically enriched context-aware oer recommendation strategy and its application to a computer science oer repository. IEEE Trans Educ 57(4):255–260
48. Said A, Jain BJ, Narr S, Plumbaum T (2012) Users and noise: The magic barrier of recommender systems. In: User modeling, Adaptation, and Personalization. Springer, pp 237–248
49. Skiena SS (2006) Revilla MA. The programming contest training manual. Springer Science & Business Media, Programming challenges
50. Sleeman D, Brown JS (1982) Intelligent tutoring systems. Academic Press, London
51. Thai-Nghe N, Drumond L, Horváth T, Nanopoulos A, Schmidt-Thieme L (2011) Matrix and tensor factorization for predicting student performance. In: Proceedings of the 3rd International Conference on Computer Supported Education (CSEDU), pp 69–78
52. Verdú E, Regueras LM, Verdú MJ, Leal JP, de Castro JP, Queirós R (2012) A distributed system for learning programming on-line. Comput Educ 58(1):1–10
53. Vesin B, Klaśnja-Milićević A, Ivanović M, Budimac Z (2013) Applying recommender systems and adaptive hypermedia for e-learning personalization. Computing and Informatics 32(3):629–659
54. Wang GP, Chen SY, Yang X, Feng R (2016) Ojpot: online judge & practice oriented teaching idea in programming courses. Eur J Eng Educ 41(3):304–319
55. Wang PY, Yang HC (2012) Using collaborative filtering to support college students' use of online forum for english learning. Comput Educ 59(2):628–637
56. Winoto P, Tang TY, McCalla GI (2012) Contexts in a paper recommendation system with collaborative filtering. The International Review of Research in Open and Distributed Learning 13(5):56–75
57. Yera Toledo R, Caballero Mota Y (2014) An e-learning collaborative filtering approach to suggest problems to solve in programming online judges. International Journal of Distance Education Technologies 12(2):51–65

58. Yera Toledo R, Caballero Mota Y, Garcia Borroto M (2013) A regularity-based preprocessing method for collaborative recommender systems. J Inf Process Syst 9(3):435–460
59. Yera Toledo R, Caballero Mota Y, Martínez L (2015) Correcting noisy ratings in collaborative recommender systems. Knowl-Based Syst 76:96–108
60. Yera Toledo R, Castro J, Martínez L (2016) A fuzzy model for managing natural noise in recommender systems. Appl Soft Comput 40:187–198
61. Yu R, Cai Z, Du X, He M, Wang Z, Yang B, Chang P (2015) The research of the recommendation algorithm in online learning. International Journal of Multimedia and Ubiquitous Engineering 10(4):71–80

Raciel Yera Toledo received the BSc. in Informatics Sciences from the University of Informatics Sciences, Cuba in 2010, his MSc degree in 2012 from the University of Ciego de Ãvila, Cuba, and his PhD degree in 2015 from Las Villas Central University, Cuba. At present, he is a professor of Computer Science at University of Ciego de Ãvila. He obtained three Annual Provincial Awards of the Cuban Academy of Sciences in 2013 for his research results related to recommender systems. Recently, he has been awarded as the Best Young Researcher at University of Ciego de Ãvila in the year 2016. His current interests are focused on the improvement of recommender systems performance through the application of computational intelligence techniques.

Luis Martínez (M'10) received the M.Sc. and Ph.D. degrees in Computer Sciences, both from the University of Granada, Spain, in 1993 and 1999, respectively. Currently, he is Full Professor of Computer Science Department and Head of ICT Research Centre at the University of Jaén. His current research interests are linguistic preference modeling, decision making, fuzzy logic based systems, computer aided learning, sensory evaluation, recommender systems and electronic commerce. He co-edited nine journal special issues on fuzzy preference modelling, soft computing, linguistic decision making and fuzzy sets theory and published more than 75 papers in journals indexed by the SCI as well as 30 book chapters and more than 120 contributions in International Conferences related to his areas. In 2015, he authored a monograph. *The 2-tuple Linguistic Model: Computing with Words in Decision Making* (Springer).

Dr. Martínez is member of the European Society for Fuzzy Logic and Technology and IEEE, Co-Editor in Chief of the of the International Journal of Computational Intelligence Systems and an Associate Editor of the journals IEEE TRANSACTIONS ON FUZZY SYSTEMS, Information Fusion, the International Journal of Fuzzy Systems, Journal of Intelligent & Fuzzy Systems, the Scientific World Journal, Journal of Fuzzy Mathematics and serves as member of the journal Editorial Board of the Journal of Universal Computer Sciences. He received twice the IEEE TRANSACTIONS ON FUZZY SYSTEMS Outstanding Paper Award 2008 and 2012 (bestowed in 2011 and 2015 respectively). He is also Visiting Professor in University of Technology Sydney, Guest Professor in the Southwest Jiaotong University and honourable professor in Xihua University both in Chengdu (China).