Appl Intell (2017) 46:328–364
DOI 10.1007/s10489-016-0830-y

A semantic-enhanced trust based recommender system using ant colony optimization

Faezeh Sadat Gohari[1] · Hassan Haghighi[1] · Fereidoon Shams Aliee[1]

Published online: 2 September 2016

Abstract Collaborative Filtering (CF) is the most popular recommendation technique that uses preferences of users in a community to make personal recommendations for other users. Despite its popularity and success, CF suffers from the data sparsity and cold-start problems. To alleviate these issues, in recent years, there has been an upsurge of interest in exploiting trust information to improve the performance of CF. In general, trust has a number of distinct properties such as asymmetry, transitivity, dynamicity and context-dependency. However, conventional trust-based CF systems do not address trust computation by considering all the properties of trust. Particularly, the context-dependency property has received less attention in the existing approaches. The consideration of all these properties leads to more accurate recommendations since the quality of the inferred is improved. In this paper, we propose a novel trust-based approach, called Semantic-enhanced Trust based Ant Recommender System (STARS), which satisfies all the properties mentioned above. Using ant colony optimization, the proposed system performs a depth-first search for the optimal trust paths in the trust network and selects the best neighbors of an active user to provide better recommendations. To consider the context-dependency property, trust inference in STARS depends on the semantic descriptions of items. Incorporation of both global and local trust in CF-based recommender systems in addition to the trust computation based on the semantic features of items allows STARS to alleviate the data sparsity, cold-start and "multiple-interests and multiple-content" problems of CF. Experimental results on real-world data sets show that STARS outperforms its counterparts in terms of prediction accuracy and recommendation quality and can overcome the above problems.

Keywords Recommender system · Collaborative filtering · Ant colony · Trust · Local trust · Global trust · Semantic similarity · Ontology

1 Introduction

The rapid increase in the amount of information over the World Wide Web has made it difficult to search and find objects that may be of interest to users. One solution to this information overload problem is the use of Recommender Systems (RSs). RSs intend to provide users with recommendations of products, services, and information they might like, taking into account their needs or preferences. In recent years, RSs have become increasingly popular and have been applied to diverse domains [1].

Two basic entities in all recommender systems are the user and the item. A user who utilizes the recommender system is called an *active user*. An active user provides her opinion about a variety of items, usually expressed in the form of ratings. The recommender system applies a filtering algorithm on the input ratings and generates suggestions about new items (i.e., target items) for the active user [2]. Collaborative Filtering (CF) is one of the most popular techniques in recommender systems. CF is the process

✉ Hassan Haghighi
h_haghighi@sbu.ac.ir

Faezeh Sadat Gohari
f_gohari@sbu.ac.ir

Fereidoon Shams Aliee
f_shams@sbu.ac.ir

[1] Faculty of Computer Science and Engineering, Shahid Beheshti University G.C., Tehran, Iran

of filtering items using the known preferences of a group of users. CF has been used fairly successfully in various domains. However, it has main limitations such as the data sparsity [3–5] and cold-start [6–9] problems. Data sparsity arises when the number of ratings obtained from users is very small compared to the number of ratings that must be predicted, and so it becomes difficult to find a significant overlap between the items rated by two users [10]. Cold-start users are new users who have provided none, or a small number of ratings. CF fails to generate reliable recommendations for cold users due to the lack of enough initial ratings.

Beside the mentioned problems discussed by many researchers, as another limitation, CF is not adaptive to environments in which users have many different interests, and at the same time, items have completely different contents. In such cases, CF provides poor recommendations because the target item for the active user may not be consistent with the common interests of her neighbors. This problem is called Multiple-Interests and Multiple-Content (denoted as MIMC) [11]. More specifically, in traditional CF methods, the similarities between users are computed based on all co-rated items, even those items that are not related to the target items. In this way, neighbors of an active user will be identical for all target items. However, a reasonable assumption is that, for different predicted items, neighbors of the same active user are likely to be different [12].

Due to the inherent problems with CF approaches, many researchers have shifted their attention to hybrid approaches that incorporate additional external information, such as demographic information [2], semantic information [10, 13], and trust information [14, 15]. More recently, semantic-based CF systems have successfully been used in different domains. Such systems can take advantages of semantic reasoning to provide much more reasonable recommendations in the case of newly added item or in very sparse data sets [16]. In recent years, semantic-based recommender systems have been applied to several different domains such as health [17], tourism/leisure [13, 18], news [19], sound/movie/music [20–22], etc. Furthermore, as a new direction, several researches have suggested that the incorporation of social trust information into the traditional CF method can resolve the data sparsity and cold-start problems and improve the quality of recommendations [14, 23–29].

In the context of recommender systems, trust can be defined as "one's belief toward others in providing accurate ratings relative to the preferences of the active user" [30]. In general, trust has a number of distinct properties [31]:

i. Asymmetry: trust is personal and subjective. More specifically, if user u trusts user v, v does not necessarily trust u.
ii. Transitivity: an important property of trust which says if user u trusts v, and v trusts p, it can be inferred that user u trusts p to some extent. This property helps to identify new neighbors for an active user by propagating trust in the network.
iii. Dynamicity: trust between two persons often changes over time. Trust can be increased with positive experiences and decreased with negative experiences.
iv. Context-dependency: trust is context-dependent, which means trust relations should be determined with respect to a particular situation. For example, a user who provides satisfying recommendations in the movie domain may not be an expert in the domain of digital cameras. The context can refer to the type of items that users give ratings or the condition in which ratings are issued, such as the location of users or items. In this paper, contextual information refers to the content descriptions of items.

Trust-based recommender systems employ trust relationships between users in a social network, known as a trust network, to produce recommendations for users based on people they trust. Within the trust network, trust can be modeled locally or globally. Global trust models compute reputation of a user within the whole network. In other words, such models estimate how the community as a whole considers a certain user. In contrast, local trust models compute a user's trustworthiness with respect to every other user [32]. Actually, local trust models derive trust values between users based on their ratings on co-rated items. In general, while local trust models can be more precise and personalized than global models, they are computationally more expensive [23]. Also, local trust models suffer from the cold-start problem [33] since it is difficult to identify trustworthy users using an insufficient number of co-rated items.

Two main trust-based filtering approaches have been adopted in the current literature: implicit trust and explicit trust. In the former approach, trust is inferred from user behaviors such as the provided ratings whereas in the latter, trust is directly specified by users [30]. Although explicit trust-based systems tend to be more accurate than the implicit ones, they require additional user effort, and thus, explicit trust statements may not always be available [24]. So, this paper focuses on the implicit trust in the proposed approach.

There are many trust inference approaches (or in other words, trust metrics) proposed to calculate the implicit trust from user ratings. Among the proposed approaches, some representative and popular trust metrics [14, 23, 25–29] have been studied by Guo et al. [31] in terms of the trust properties. Their study reveals that these metrics are not asymmetric since they derive trust values based

on similarity or error measures which are symmetric in nature. Although all these metrics [14, 23, 25–29] are transitive, none of them explicitly consider the dynamicity and context-dependency properties of trust. In summary, the proposed approaches partially cover trust properties, and new metrics are needed to better satisfy the semantics of trust [31]. Among the proposed approaches, the work of Shambour and Lu's [14] is relatively close to ours since it fuses the trust and semantic information of users and items within the CF framework. They proposed an innovative Trust–Semantic Fusion (TSF)-based recommendation approach, which merges two hybrid recommendation approaches: the user-based trust-enhanced CF and the item-based semantic-enhanced CF. The former approach utilizes trust information to alleviate the data sparsity and cold-start problems, whereas the latter approach employs the semantic features of items to address these problems. Therefore, in the TSF approach, two separate modules use the trust information and semantic information, and thus, trust values are computed independently of semantic features of items. In contrast, in our proposed approach, trust inference relies on the items' semantic descriptions in order to handle the context-dependency of trust. It is remarkable that TSF only satisfies the transitivity property [31]. As will be shown in our experiments, ignoring the main properties of trust in TSF results in predictions with less accuracy and recommendations with lower quality compared to our approach.

It has been recently shown that by applying Ant Colony Optimization (ACO) algorithms [34, 35] to trust-based recommender systems, it is possible to handle the dynamicity of trust [36]. Ant colony algorithms are a group of meta-heuristic optimization algorithms based on the ants' efforts for seeking food in nature. These algorithms, which utilize random procedures and reinforcement learning, are extremely satisfactory in dynamic environments. Ant colony algorithms are based on emulation of behavior of real ants. In nature, real ants aim to find the optimal path between a food source and their nest without direct communications, adapting to changes in the environment. One factor that the ants benefit from is pheromone deposition. Ants are attracted by pheromones coming from fellow type ants. As the time passes, paths with higher pheromone levels are chosen with a higher probability than those that have a weaker amount of pheromone deposit. This collaborative behavior between fellow type ants is similar to the collaborative world as people mostly collect opinions from their like-minded friends (or neighbors) [36].

In the context of trust-based CF recommender systems, ACO can be applied on a directed trust graph in order to search optimal trust paths. Actually, ACO selects the paths with the maximal propagated trust values as the most trustworthy paths. These trustworthy paths help to identify the best neighbors of an active user because a chain of users with high propagated trust values can provide more precise opinions for the active user. To be more specific, the active user is considered as the ants' nest and her trusted neighbors as the food sources. So, the artificial ants are dispatched from the active user node into the trust network to imitate the foraging behavior of real ants in search for a valuable food source. Dynamicity of trust, as an important property for improving the quality of recommendations, can be effectively handled using the pheromone updating strategy of ants to analyze the trust intensity among users over time. Based on this strategy, the pheromone value associated with trustworthy neighbors is increasing, while this value for the other users is decreasing. The application of ACO to the area of implicit trust-based filtering approaches has been studied in the Bedi and Sharma's work [36] where they addressed time-based trust computation using the pheromone updating strategy. They proposed the Trust based Ant Recommender System (TARS) which produces valuable recommendations by incorporating the notion of dynamic trust between users and selecting the best neighborhood based on the biological metaphor of ant colonies. As time passes, TARS produces recommendations by continuously updating the dynamic trust between users. TARS satisfies asymmetry, transitivity and dynamicity properties but not context-dependency property. It also has some other limitations that will be briefly explained in Section 2.3.

In conclusion, the literature on trust-based CF systems reveals the absence of a recommender system that would take into account all the properties of trust, especially, context-dependency which has not yet been investigated by any of the previous implicit trust-based filtering approaches in an empirical manner. In order to fill the current gap, we extend TARS developed by Bedi and Sharma [36] and propose STARS (Semantic-enhanced Trust based Ant Recommender System) which satisfies all the properties of trust. Using semantic information and clustering items based on their semantic similarities, STARS incorporates the context information in trust computation. More specifically, for each item cluster c, STARS creates a directed Implicit Trust Graph (ITG). An ITG^c is a directed graph where the nodes are users (So, we use the words "node" and "user" interchangeably in the paper). The edges are weighted according to the degree of trust from one user to another user in the context of item cluster c. In order to find the best neighbors of an active user for target item x, STARS applies ACO on ITG^{ε}, where ε refers to the cluster that item x belongs to. Thus, implicit trust values between users depend on the semantic features of the target item. In other words, for different target items, trusted neighbors of the same active user may be different.

STARS works in two phases: offline and online. Starting at time $t = t_0$, it creates a set of initial ITGs based on each item cluster in the offline mode. In order to create initial ITGs, STARS uses a global trust model. In the online phase, STARS first determines which semantic cluster the target item belongs to, and then uses the ITG associated with that cluster to implement ACO for selecting the best neighbors of the active user. In order to control the maximum search depth, STARS uses a tunable trust propagation limit. After neighborhood formation, STARS predicts the active user's ratings for target items and produces a list of recommendations. Finally, using the ACO pheromone updating strategy, STARS updates each ITG such that the pheromone increases for the trustworthy neighbors and decreases for other users. It should be noted that the updating step is also accomplished offline.

In comparison with research efforts found in the literature, our work has the following differences:

- A novel implicit trust-based filtering approach, called STARS, which satisfies all the distinct properties of trust, especially, context-dependency. To the best of our knowledge, this is the first trust-based CF system that is asymmetric, transitive, dynamic and is able to leverage context information in trust computation.
- A novel ant-inspired search algorithm for finding the best neighbors of an active user with respect to a specific target item x at each time step. This algorithm utilizes both trust and similarity information in the context of cluster ε that item x belongs to. Trust and similarity information is represented by pheromone levels on the edges and heuristic values of the nodes in ITG^{ε}, respectively.
- A novel ant-inspired algorithm for updating the dynamic trust (through pheromone evaporation and deposition) between the active user and other users in ITG^{ε} at each time step. In this algorithm, the amount of pheromone to be deposited depends on: (1) the inferred trust values through the best trust paths in ITG^{ε}, and (2) the amount of confidence which is directly related to the number of co-rated items between two users.
- A new method for considering both global and local trusts in a recommender system. More specifically, STARS uses global trust at the initial stage (time $t = t_0$), and as the time passes (time $t > t_0$), it locally updates the dynamic trust value between the active user and others. Using the global trust at the initial stage, the system can provide reasonable recommendations for cold users with a few or even no ratings. As the time passes and the active user provides more information about her preferences (e.g., rates an item x), STARS locally adjusts the weight of her outgoing links in ITG^{ε} so that her most trusted neighbors have a higher probability of selection in future.

Incorporation of both the global and local trusts into CF along with the trust computation based on the semantic features of items contributes to the success of STARS in alleviating the data sparsity, cold-start and MIMC problems of CF. An exhaustive set of temporal experiments on the Movielens data sets empirically evaluated the performance of the proposed approach over time and demonstrated its advantages over benchmark algorithms. In order to incorporate time into our experiments, we used rating timestamp available in the Movielens data sets.

The rest of this paper is organized as follows. The related work is reviewed in the following section. In Section 3, the STARS approach and its components are elaborated. Section 4 demonstrates the experiments and their results. Finally, we present our conclusions and outline the future lines of research in Section 5.

2 Background

In this section, the related subjects to the proposed recommender system are explained. First, a short introduction about ACO is presented. Then, we review the literature related to the CF recommender systems and the hybrid systems. The studied hybrid systems exploit additional sources of knowledge (such as trust relations between users and/or semantic features of items) to improve the performance of CF.

2.1 Ant colony optimization

Swarm intelligence is a computational and behavioral metaphor for problem solving that takes inspiration from the social behavior of insects or other animals. ACO is one of the most powerful optimization methods that takes inspiration from the foraging behavior of real ants [37]. In nature, ants deposit pheromone on the ground in order to communicate each other. The deposited pheromone helps the ants to find the shortest path between the nest and the food. More specifically, in searching for a food source, ants smell the pheromone left by previous ants of the same colony and tend to follow the paths marked by strong pheromone concentrations. In other words, ants choose their path by a probabilistic decision guided by the amount of pheromone: the larger the amount of pheromone on a trail, the higher the probability that ants follow that trail when choosing their path.

During the return trip, the amount of pheromone deposited on the trail depends on the quantity and quality of

the food source. This indirect communication among ants helps them to find the shortest path. Since the shorter paths take less time to be traversed, they are reinforced more with the greater amount of pheromone. Therefore, the shorter paths become more favored over time. It should be noted that the pheromone gradually evaporates. Thus, the longer paths lose their pheromone intensity and become less attractive over time. The final result is that the majority of the ants will quickly trace the shortest path between the nest and a food source [38–40].

Various ACO algorithms exploit a similar mechanism for solving optimization problems. Ant colony problems are usually modeled with a decision graph [37]. In an ACO algorithm, each artificial ant constructs a candidate solution by a sequence of probabilistic decisions. The decisions are biased by the amount of pheromone deposited on the edges, and available heuristic information. The sequence of decisions for constructing a solution can be viewed as a path through the corresponding decision graph. Finding good solutions for the problem is done in an iterative process. In each iteration, the solutions found by the ants guide the process of solution construction in the following iterations. More specifically, in each iteration, the pheromone trails are updated to guide the ants towards paths that are more likely to result in good solutions. This process continues until some stopping criterion is met. For example, stopping criteria could include reaching a maximum number of iterations or finding a solution of a given quality [41].

2.2 CF recommender systems

CF is one of the most popular techniques in recommender systems [14]. CF approaches are divided into two categories: memory-based approaches and model-based approaches. In memory-base approaches, the entire rating matrix is used to make recommendations. In model-base approaches, a model is driven from the previous ratings. Then, the driven model is used to make the predictions [42]. Memory-based approaches can be further classified into two main classes: User-based CF (UCF) [43] and Item-based CF (ICF) [44]. In UCF approaches, a subset of users is chosen based on their similarity to the active user (this subset is called the *neighborhood*). Then, a weighted combination of the neighbors' ratings is used to predict the ratings for the active user. ICF approaches are similar to UCF approaches, except that the former ones employ the similarity between the items instead of users [45]. Despite the popularity of CF approaches, they suffer from data sparsity, cold-start and MIMC problems [11, 14, 30]. Among these problems, MIMC has received less attention in the existing works. This problem is occurred when users are interested in a variety of items that have different content. In such cases, CF cannot provide accurate recommendations because the target item for the active user may not be consistent with the common interests of her neighbors [11]. MIMC problem can be alleviated by considering the similarity between items when finding neighbors of an active user. For example, Li et al. [11] proposed a hybrid approach by integrating both UCF and ICF methods. This approach is able to filter the dissimilar items to the target item and select neighbors of the active user based on the similar items to the target. In [12], a new similarity function has been proposed to select neighbors who are more appropriate according to each specific target item. In the proposed function, the rating of a user on an item is weighted based on the similarity between this item and the target item.

Many successful approaches have been developed over the past few years to alleviate the data sparsity and cold-start problems. These approaches explained in the subsequent sections usually rely on additional sources of knowledge, such as items' semantic descriptions [10, 13] and/or users' trust information [14, 15].

2.3 Semantic-based CF recommender systems

In recent years, reasoning techniques borrowed from the Semantic Web have been adopted in the context of recommender systems in order to overcome the data sparsity and cold-start problems [10]. Traditional syntactic-based recommender systems miss a lot of useful knowledge during the recommendation process. Therefore, their recommendations only include items very similar to those the user already knows. Semantic-based recommender systems can overcome this problem by inferring implicit semantic relationships between items [46]. The cornerstone of the Semantic Web is the use of taxonomies or ontologies to classify and describe concepts in a particular domain. Using product taxonomies and ontologies allows the system to reason about the semantics of items and to discover the hidden semantic associations between them [10]. Semantic-based CF approaches [10, 13, 20, 47, 48] provide two primary advantages over traditional CF approaches. First, the semantic attributes of items allow the system to make inferences based on the underlying reasons for which a user may or may not be interested in a particular item. Second, in the case of a new item or in very sparse data sets, the system can still use the semantic information to provide reasonable recommendations for users [16].

2.4 Trust-based CF recommender systems

As another approach to alleviate the problems of traditional CF approaches (such as the data sparsity and cold-start), trust information has been widely used in CF methods.

The resulting hybrid systems typically explore the trust network and find a neighborhood of users trusted (directly or indirectly) by a user and generate recommendations by aggregating their ratings [15]. As mentioned earlier, there are two main trust-based filtering methods: explicit trust and implicit trust. Since in explicit approaches trust values are obtained from pre-existing social links between users, asymmetry of the trust is always satisfied. In contrast, implicit approaches are often symmetric since they are based on the similarity or error measures which are symmetric in general [31]. According to [49], the performance of using implicit trust information is slightly worse than applying explicit trust information. Nevertheless, explicit trust-based filtering approaches encounter two major limitations [14]: (1) they require extra user efforts to label the trust statements. Accordingly, explicit trust statements may not always be available; (2) they suffer from the cold-start problem since new users should specify explicitly whom they trust before the filtering becomes effective. These limitations make the implicit trust-based filtering approaches more feasible to use [14]. Hence, the present work focuses on the implicit trust.

Most of the trust-based recommendation approaches use the transitivity of trust and propagate trust to indirect neighbors in the social network. However, the dynamicity and context-dependency of the trust have been often ignored in the proposed approaches. Here, we review some major and popular implicit trust-based filtering approaches. Due to space restrictions, we only present related works on the implicit trust. To the best of our knowledge, there is not any explicit trust-based recommender system that takes all distinct properties of trust into account.

Implicit trust-based filtering approaches derive trust values based on users' ratings on items. For instance, O'Donovan and Smyth [28] define the "profile-level" and "item-level" trust as the percentage of correct predictions that a profile has made "in general" or "with respect to a particular item", respectively. Pitsilis and Marshall [29] proposed a model of implicit derivation of the user's trust from an evidence that describes her rating behavior. In the proposed model, trust is expressed in the form of opinion which is always subjective and uncertain. Papagelis et al. [26] developed a trust computational model permitting to consider the subjective notion of trust associations by applying confidence and uncertainty properties. Lathia et al. [25] proposed a trusted k-nearest recommenders algorithm allowing users to learn how much to trust one another by evaluating the utility of the rating information they receive. Hwang and Chen [23] proposed an implicit trust metric deriving trust scores directly from the ratings data based on the users' prediction accuracy in the past. Yuan et al confirmed the small-world property of the trust network [50] and developed an implicit TrustAware Recommender System (iTARS) [27] based on this property. This property indicates that the trust propagation distance between any two randomly selected users of the trust network is short. Shambour and Lu [14] proposed the TSF recommendation approach providing more effective results, in terms of prediction accuracy and coverage, compared to the user-based and item-based benchmark algorithms. As mentioned before, this approach fuses the "user-based trust-enhanced CF" with the "item-based semantic-enhanced CF" approaches. The previously mentioned implicit trust metrics do not handle dynamicity, context-dependency and asymmetry [31]. The recent work conducted by Fang et al. [51] takes into account the context-dependency of trust. The authors focused on predicting implicit trust and distrust values according to interpersonal and impersonal aspects of trust and distrust. In their proposed model, interpersonal aspects are computationally modeled based on users' historical ratings, while impersonal aspects are computed on the basis of users' explicit trust and distrust network. In this model, competence, as one of the interpersonal trust aspects, is computed under a specific context. In other words, the user receiving a high competence belief from the trustor is capable of providing satisfactory recommendations to the trustor in a specific context. However, the authors did not take into account the context information in their experiments. Also, the dynamic aspect of trust was not considered in their model.

It has been recently shown that by applying ACO to trust-based recommender systems, it is possible to handle the dynamicity of trust [36, 52, 53]. This property can be viewed as the trust intensity between users. The trust intensity between an active user and each of her neighbors may change depending on recommendations generated by the neighbor. The pheromone updating strategy in ACO algorithms can be used to analyze the trust intensity among users over time. Based on this strategy, the pheromone value associated with trustworthy neighbors is increasing, while this value for the other users is decreasing [36]. The first successful application of ACO to the context of trust-enhanced recommender systems is T-BAR (Trust-Based Ant Recommender) [52]. T-BAR is a dynamic algorithm based on the probabilistic model of ACO algorithms, which its ability to increase the accuracy and coverage of predictions has been proven. In T-BAR, explicitly expressed trust values are used as heuristic information. In addition, using the trust information in the network, the pheromone level of each edge is locally initialized before an ant encounters it. However, this algorithm suffers from its inability to deal with cold users. To overcome this problem, the authors proposed DT-BAR (Dynamic T-BAR) [53], a dynamic trust-based recommender that solves the new user problem by allowing the ants to share information about the traversed edges.

The application of ACO to the context of implicit trust-based filtering approaches has been studied in TARS [36]. TARS produces valuable recommendations by incorporating a notion of dynamic trust between users and selecting the best neighborhood based on the biological metaphor of ant colonies. In TARS, the pheromone level on edges represents the strength of connectedness, i.e., the trust intensity between the two recommendation partners; also, heuristic values depend on the level of connectedness from the active user node to another node chosen by ants. The initial pheromone level is computed by combining similarity and confidence measures. The combination of similarity with confidence reduces data sparsity and creates asymmetric trust values. As the time passes, TARS produces recommendations by continuously updating the dynamic trust between users.

Nevertheless, the major limitation of TARS stems from the fact that it performs a modified breadth-first search in the trust network to generate recommendations. More specifically, TARS initially selects the best neighborhood only among the direct neighbors (i.e., users at distance 1 from the active user). The active user's rating for an item is generated by aggregating the ratings of selected neighbors for that item. If some items are not rated by any of the direct neighbors, then TARS moves to the next level and explores the child nodes of the most trustworthy user. This process is repeated until the ratings of all unrated items are predicted. So, due to the proceeding of the search in a breadth-first manner, if direct neighbors have rated an item, then TARS is unable to use possible valuable ratings of the users without direct trust link to the active user. The other limitation of this approach is that the use of distance metric in computing heuristic values would not be effective in distinguishing between trusted and untrusted friends in a breath-first search process. Also, TARS does not take into account changes in user interests over time. Actually, it considers user interest profiles only during the initial stage (i.e., at time $t = t_0$) in which it uses similarity between profiles for computing the initial trust values. After that, at time $t > t_0$, it does not consider new or changed interests of the active user; it updates the trust information only based on the involvement or non-involvement of other users as a recommender for the active user over a period of time. As the final shortcoming, TARS does not take into account the context information in its trust model.

As seen, the literature on trust-based CF systems reveals the absence of a recommender system that takes all distinct properties of trust into account. In order to fill this gap, we propose a novel implicit trust-based filtering approach, called STARS, which is asymmetric, transitive, dynamic and is able to leverage the context information in trust computation. Using the ant colony optimization, STARS performs a depth-first search for the optimal trust paths in the trust network and selects the best neighbors of the active user. In this approach, trust can be passed from one member to another in the trust network, creating trust chains, based on the propagative and transitive nature of the trust. STARS considers the contextual information by inferring trust values based on the semantic descriptions of items. This approach also handles the asymmetry and dynamicity properties using the pheromone updating strategy. Based on this strategy, at each time step, the pheromone value associated with the best neighbors of the active user in a specific context is increased, while this value for the other users is decreased. As will be shown in the following sections, STARS adapts itself to dynamically changing user interests and mitigates the data sparsity, cold-start and MIMC issues.

3 STARS: semantic-enhanced trust based ant recommender system

STARS is a dynamic recommender system which considers contextual (i.e., content descriptions of items) and temporal information for selecting the most trustworthy neighbors of the active user according to her current interests in specific types of items. To infer context-dependent trust values, STARS uses the semantic descriptions of items. Actually, STARS clusters items based on their semantic similarities and finds trusted neighbors of an active user with respect to a specific cluster. Let $I = \{i_1, i_2, ..., i_m\},\ m > 1$, be a given set of items and $U = \{u_1, u_2, ..., u_n\},\ n > 1$, be a given set of users. By clustering items based on their semantic similarities, we have a set of z item clusters $C = \{c_1, c_2, ..., c_z\},\ z > 1$, such that each cluster c_j contains at least one item (i.e., $|c_j| \geq 1$, where $|c_j|$ denotes the number of items in cluster c_j). Each cluster represents a specific context. In the rest of the paper, wherever we mention a context c, it refers to cluster c.

Temporal information is incorporated into STARS by exploiting timestamps of ratings. In other words, STARS observes the ratings over time. Let $\mathbf{R}(t)$ be an $n \times m$ user-item rating matrix which stores the ratings made on and before time slot t. Each element $r_{u,i}$ of this matrix represents the rating of item i by user u. Besides, $r_{u,i}$ is associated with a timestamp. Let matrix $\mathbf{S}(t)$ with the same size of $\mathbf{R}(t)$ involve the corresponding timestamp of ratings. Each element $t_{u,i}$ of matrix $\mathbf{S}(t)$ is the timestamp of the rating made by user u on item i. So, as a dynamic recommender system, STARS continuously collects users' feedbacks over a long period of time. In this paper, we use time values with day granularity. Starting at time $t_0 = x$ (i.e., x days from the first rating), STARS is iteratively updated at every μ days. So, the length of the first time slot (i.e., time slot t_0) is equal to x days, while the length of other time slots is equal to μ days. At each time step t, the task of STARS is to predict ratings

of the active user for target items at time slot $t+1$, produce a list of top-N recommendations, and update dynamic trust values for the next time slot.

As mentioned before, STARS leverages context information in neighborhood formation at time slot t by considering users' interest in each item cluster. In other words, STARS infers multiple trust relationships between two users, each of which corresponds to a specific context. For this purpose, STARS splits the rating matrix $\mathbf{R}$ and timestamp matrix $\mathbf{S}$ into z sub-matrices, each of which corresponds to a cluster of items. Considering cluster c_j, STARS produces an $n \times |c_j|$ sub-matrix, called $\mathbf{R}^{c_j}(t)$, which contains user ratings for all items in cluster c_j up to time t. It also produces a sub-matrix $\mathbf{S}^{c_j}(t)$ with the same size of $\mathbf{R}^{c_j}(t)$, which contains the corresponding timestamp of ratings. At the initial time slot t_0, STARS uses $\mathbf{R}^{c_j}(t_0)$ in order to compute the initial global trust values between users in the context of cluster c_j. Using a global trust model at the initial stage, users can benefit from opinions of globally trusted users. This is especially useful for the cold users who have only rated a small number of items. As the time passes and users provide information about their preferences for items belonging to cluster c_j, STARS uses $\mathbf{R}^{c_j}(t)$ and $\mathbf{S}^{c_j}(t)$ to locally update the dynamic trust values between users in this context.

Using the ACO algorithm, STARS applies the following information for selecting the most trustworthy neighbors of the active user in a specific context c at each time slot:

1) The previously learned trust knowledge based on the global reputation and involvement of users in generating recommendations in context c in the past. This knowledge is memorized in the form of pheromone trails. Based on this knowledge, users who have been selected more frequently as the trusted neighbors of an active user have a higher probability of selection compared to other users.
2) The similarity between users according to the current interests of users in items belonging to context c. This knowledge is represented as the heuristic value of nodes (users) in a trust network during the neighborhood selection process. In order to compute the similarities between users, STARS exploits timestamps of ratings. The reason is that STARS is a dynamic recommender system, and thus, the existing users' preferences are not static and may change over time. With respect to the fact that implicit trust is positively correlated with user interests, the changes in users' preferences affect the users' tendencies to trust or not to trust others. To be able to adapt to such changes, when the recommendations are requested, STARS computes similarity between users using a temporal relevance measure [54]. This measure gives more importance to recent observations and reduces the effect of old ratings since the recent ratings could better reflect the user's current interest [54]. The utilization of the time-based similarity information (as the heuristic values) helps STARS to select more accurate neighbors and better update the trust pheromone values according to the users' current preferences. So, STARS is able to adapt to dynamically changing user interests and select the most trustworthy neighbors of the active user according to her current interests in specific types of items.

In the following sections, the architecture of STARS is described, and each of its components is discussed in detail. Then, the computational complexity of STARS is analyzed.

3.1 The architecture of STARS

As shown in Fig. 1, STARS contains two main components: database and recommendation engine. The first component involves the development and storage of the item ontology and data structures. The item ontology helps to classify, describe and interrelate the universe of relevant concepts in a specific domain [10]. The second component generates a list of top-N recommendations for users. It enables the system to reason about the semantic descriptions of the available items and to infer hidden semantic relationships between them [46]. The recommendation engine has three modules, namely preprocessor, recommender and trust network updater. Initially, at time $t = t_0$, the preprocessor module, which works in the offline mode, computes the similarity between every two items based on their semantic descriptions, as given in the item ontology. Then, the preprocessor module clusters the items according to their semantic similarities by the k-medoids algorithm [55]. Finally, this module creates a set of initial ITGs based on the global trust values between users in each item cluster. In the online mode, at each time slot t, the recommender module first retrieves context-specific data (i.e., trust and rating data) and implements ACO for selecting the best neighbors of the active user in each context. After neighborhood formation, it predicts unknown ratings of the active user and produces a list of recommendations. Finally, the third module uses the ACO pheromone updating strategy to update the ITG related to each context. The third module works offline. As the result of this update, the pheromone increases for the trustworthy neighbors and decreases for other users in that context. In the following sections, we describe each module of the recommendation engine in detail.

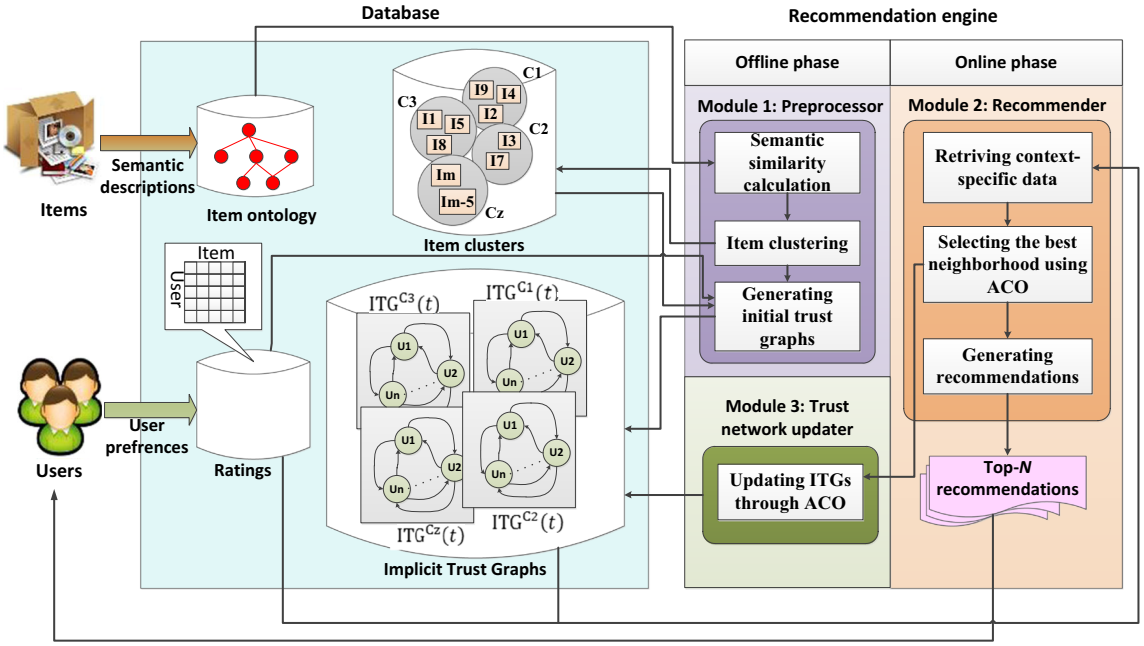


Fig. 1 The architecture of the STARS approach

3.1.1 Preprocessor module

The aim of this module is to incorporate the context information into STARS using semantic profiles of items. For this purpose, this module first generates item clusters based on their semantic similarities and then creates an ITG for each cluster. The main intuition behind this idea is that the trust value between two users depends on the content of the existing items. For example, a user who provides valuable recommendations for purchasing cars may not be an expert in the movie domain. In other words, a trustworthy user is capable of providing satisfactory recommendations to the trustor only in specific contexts [51].

Calculating implicit trust values based on the semantic features of items makes STARS adaptive to multiple-interests environments. Actually, the implicit trust is positively correlated with user interests. Thus, when users have many different interests, and items have completely different content, multiple trust relationships may exist between users. Each of these trust relationships corresponds to a different context. By applying context-dependent trust relationships, selected neighbors of a specific active user may be different with respect to different target items, which results in alleviating the MIMC problem.

In order to create initial ITGs, the preprocessor module uses a global trust model. Global trust models predict a global "reputation" score that estimates how the community as a whole considers a certain user [32]. The application of the global trust model at the initial stage helps to provide more neighbors for every single user. This method is useful when there is not enough rating information (such as in the case of new users or very sparse data sets).

Semantic similarity calculation In order to utilize the semantic information of items, we first have to represent item characteristics with a domain ontology. The item ontology implemented in our system includes the typical concepts and relationships of the movie domain. Although our approach can potentially be used for other domains with different item ontologies, it has been implemented in the movie domain because it is a well-known domain and there are a large number of relationships between the concepts involved in this domain (such as movies, actors, directors, writers, genres, etc.). We use the Movie Ontology[1] which has been developed according to the Ontology Web Language (OWL) standard by the University of Zurich. The Movie Ontology provides a controlled vocabulary to semantically describe movie related concepts (such as Actor, Director, Genre, etc.) and their associated individuals. Through this ontology, it is possible to link, hierarchically and non-hierarchically, elements belonging to the domain of the movies. The main class of the ontology is class "Movie". All the movies are instances of this class. For instantiating the Movie Ontology,

[1] http://www.movieontology.org/

we use the Internet Movie Database[2] and gather required data using a web crawler.

The similarity between two items a and b is computed based on their semantic descriptions, as given in the item ontology. For this purpose, we use the semantic similarity formula presented by Carrer-Neto et al. [56]:

$$Semsim(a,b) = \sum_{i=1}^{|P|} \left(\frac{\text{common}(a,b,P[i])}{\max(\deg(a,P[i]),\deg(b,P[i]))} \right) \times Weight(P[i]) \quad (1)$$

where P is a vector that contains a set of datatype properties and object properties of the Movie class representing the "target" of the recommender engine; $\deg(a,p)$ represents the number of instances associated with item a through property p; $\text{common}(a,b,p)$ denotes the number of common instances associated with items a and b through property p; and $\text{Weight}(p)$ indicates the importance of property p. The weights of the properties should be determined subjectively according to the given domain. For example, in the movie domain, the Genre of a movie is more important than its filming locations.

The main datatype and object properties that are used in our system are as follows: *belongsToGenre*, *hasActor*, *hasDirector*, *hasProducer*, *isAwardedWith*, *nominatedFor*, *hasFilmLocation*, *isProducedBy*, and *isFromDecade*. For detailed information about these properties, refer to Carrer-Neto et al. [56] (Section 3.1.1).

Item clustering For semantic clustering of items, we have adopted the k-medoids algorithm [55] due to its simplicity and high accuracy. Similar to the k-means algorithm [57], k-medoids is a partition based clustering algorithm. However, in contrast to k-means, k-medoids chooses objects as centers (medoids) instead of taking the mean value of the objects, and it can work with an arbitrary matrix of distances between objects. Since the k-medoids algorithm minimizes the summation of pairwise distances within a cluster, it is more robust to the noise and outliers compared to the k-means algorithm. Moreover, k-medoids is not generally influenced by the presentation order of objects. In this paper, we have adopted the k-medoids algorithm in order to preserve items' semantic information during the clustering process described as follows.

First, the k-medoids algorithm randomly selects k items as the initial medoids. Then, it proceeds by alternating between two steps. In the first step, each item is assigned to the cluster associated with the nearest medoid. In particular, the semantic similarity is used as the distance metric to measure the closeness of two items. The item distance between two items a and b, denoted by $dist(a,b)$, is computed by $dist(a,b) = 1 - Semsim(a,b)$. In the next step, within each cluster, each of the non-medoid items is swapped with the medoid. If the sum of within-cluster distances decreases using a non-medoid item as the medoid, then that item is chosen as a new medoid. The algorithm repeats these steps until the medoids become fixed.

Generating initial trust graphs After generating item clusters, the preprocessor module creates a directed implicit trust graph for each cluster. The implicit trust graph $\text{ITG}^c(t)$ is a directed graph where the nodes are users and the edges are implicit trust relationships. In this graph, the edges are weighted according to the degree of the trust between each pair of users at time slot t, considering only items belonging to cluster c. In order to create initial ITGs, a global trust model is used. A user's global trust can be computed as the average of the local trust scores given by direct neighbors of the user [23].

As described before, STARS derives the implicit trust values from context-related data. At time $t = t_0$, the initial $\text{ITG}^{c_j}(t_0)$ is created based on the available ratings in submatrix $\mathbf{R}^{c_j}(t_0)$. Each $\text{ITG}^{c_j}(t_0)$, $j \in 1, 2, ..., z$, is created as the following steps:

Step 1: *Normalize rating data*

Because ratings are determined not only by user interests but also by rating habits of users, it is important to normalize ratings of different users to the same scale [58]. In STARS, user ratings are normalized in the range [0,1] using the Min–Max Normalization method. Here, this method linearly transforms an original rating value r of data set X to a new value r' which fits in the range [0,1] as follows:

$$r' = \frac{r - \min(X)}{\max(X) - \min(X)} \quad (2)$$

where min(X) and max(X) denote the minimum and the maximum value of ratings in data set X, respectively.

Step 2: *Compute the initial local trust*

Local trust models consider the personal and subjective views of the users and predict personalized trust scores from each single user's point of view [32]. In this step, STARS uses the rating matrix and calculates the direct implicit trust score of every pair of users. In particular, STARS derives the local trust score by averaging the prediction error on co-rated items. Therefore, if two users have no co-rated item, then there is no direct trust relationship between them. STARS modifies the metric proposed by Shambour and Lu [14] for computing the implicit trust values. This metric [14] uses the Mean Squared Differences (MSD) method [59] to measure the degree of trust between two users by averaging the prediction error on co-rated items. Also, in this metric,

[2] http://www.imdb.com/

the proportion between the common ratings and the total rated items are taken into consideration to derive the implicit trust values [14].

Let $I_u^{c_j}(t_0)$ be the set of items in cluster c_j rated by user u until time t_0, and $\left|I_u^{c_j}(t_0)\right|$ represents the number of elements in set $I_u^{c_j}(t_0)$. According to the available normalized ratings in sub-matrix $\mathbf{R}^{c_j}(t_0)$, the direct trust score assigned by user u to user v in the context of cluster c_j at time $t = t_0$, $trust_{u\to v}^{c_j}(t_0) \in [0, 1]$, is computed as:

$$trust_{u\to v}^{c_j}(t_0) = \left(1 - \frac{\sum_{i\in D\cap E}(p_{u,i}^v - r_{u,i})^2}{|D\cap E|}\right) \times \frac{|D\cap E|}{|D| + |E| - |D\cap E|}, \quad D = I_u^{c_j}(t_0), \quad E = I_v^{c_j}(t_0) \tag{3}$$

where $D \cap E$ is the set of items in cluster c_j that have been commonly rated by both users u and v; and $p_{u,i}^v$ is the predicted rating of item i for user u by only considering neighborhood user v; $p_{u,i}^v$ is calculated by:

$$p_{u,i}^v = \overline{r_u} + (r_{v,i} - \overline{r_v}) \tag{4}$$

where $\overline{r_u}$ and $\overline{r_v}$ are the mean ratings of users u and v for items belonging to cluster c_j, respectively.

Step 3: Compute the initial global trust

In this step, STARS computes the global trust score of each user v as the average of the local trust scores given by direct neighbors of this user in the trust network [23]. So, in the context of cluster c_j, the global trust score of user v at time $t = t_0$, called $Gtrust_v^{c_j}(t_0) \in [0, 1]$, is:

$$Gtrust_v^{c_j}(t_0) = \frac{1}{\left|NB_v^{c_j}\right|} \sum_{u\in NB_v^{c_j}} trust_{u\to v}^{c_j}(t_0) \tag{5}$$

where $NB_v^{c_j}$ is the set of direct neighbors of user v in context c_j.

Step 4: Create the initial directed ITG

In this step, the initial $\text{ITG}^{c_j} = (V, E)$ is created. V is the set of vertices correspond to the users and E is the set of edges connecting users. The weights on incoming links to each node v are equal to the global trust score of that node:

$$\forall u \in U,\ u \neq v \quad W_{uv}^{c_j}(t_0) = Gtrust_v^{c_j}(t_0) \tag{6}$$

where $W_{uv}^{c_j}(t_0)$ denotes the weight of the link from node u to node v in $\text{ITG}^{c_j}(t_0)$. Therefore, at the beginning, STARS predicts the same value of trustworthiness of user v for every user. As the time passes and more data is collected about the preference of the users for items belonging to cluster c_j, STARS locally adjusts the weight of corresponding links in this graph.

3.1.2 Recommender module

This module is in charge of retrieving context-specific data, analyzing the retrieved data to select the best neighborhoods using the ant colony metaphor, and suggesting matching items that users might like. In the following subsections, the process of this module is described in detail.

Retrieving context-specific data In the online phase, in order to create a list of top-N recommendations for active user u, the appropriate trust and rating data are first retrieved from the database. Let $\tilde{I}_u(t) = \{i \mid i \in I \text{ and } r_{u,i} = null\}$ be a set of target items which user u had not rated until time slot t, and $\tilde{C}_u(t) = \{c \mid c \in C \text{ and } (\tilde{I}_u(t) \cap c) \neq \emptyset\}$ be a set of target clusters each of which contains at least one target item $i \in \tilde{I}_u(t)$. The context-related data for user u at time slot t is as follows:

(1) $T_u(t) = \{\text{ITG}^c(t) \mid c \in \tilde{C}_u(t)\}$, a set of trust graphs required for generating recommendations for active user u at time slot t such that each graph corresponds to a target cluster c.

(2) $R_u(t) = \{\mathbf{R}^c(t) \mid c \in \tilde{C}_u(t)\}$, a set of rating sub-matrices required for generating recommendations for active user u at time slot t such that a sub-matrix $\mathbf{R}^c(t)$ contains available normalized user ratings, up to time t, for items in a target cluster c.

(3) $S_u(t) = \{\mathbf{S}^c(t) \mid c \in \tilde{C}_u(t)\}$, a set of timestamp sub-matrices required for generating recommendations for active user u at time t such that a sub-matrix $\mathbf{S}^c(t)$ contains corresponding timestamps of ratings in $\mathbf{R}^c(t)$.

This contextual data is the key input to our ant-inspired neighborhood selection algorithm as detailed in the next subsection.

Selecting the best neighborhood using ACO In this step, the recommender module selects the best neighborhood of active user u for each target item $i \in \tilde{I}_u(t)$. For this purpose, STARS uses a novel ant-inspired search algorithm which performs a depth-first search in the trust network to select the best neighbors of the active user. In the proposed algorithm, trust can be passed from one member to another in a trust network, creating trust chains, based on its propagative and transitive nature. In each context $c \in \tilde{C}_u(t)$, the trust value between active user u and each of the other users is iteratively learned and memorized as the pheromone value of the edge connecting their nodes in $\text{ITG}^c(t)$. During the search process, the heuristic value of a node depends on the similarity between this node and the active user node. This similarity is calculated with respect to users' current interests in context c. To place more emphasis on the recent

Algorithm 1 Ant-inspired neighborhood selection for active user u at time t

Inputs:

- $U = \{u_1, u_2, ..., u_n\}$, $n > 1$—the user set;
- $\tilde{I}_u(t)$ —the set of target items for user u at time t;
- $\tilde{C}_u(t)$ —the corresponding target clusters of $\tilde{I}_u(t)$;
- $T_u(t)$ —the set of implicit trust graphs at time t, according to the target clusters $\tilde{C}_u(t)$;
- $R_u(t)$ —the set of rating sub-matrices at time t, according to the target clusters $\tilde{C}_u(t)$;
- $S_u(t)$ —the set of timestamp sub-matrices at time t, according to the target clusters $\tilde{C}_u(t)$;
- θ —time decay weight;
- φ —the maximum trust propagation distance;
- bpn—the maximal number of the best trust paths;
- α —relative importance of the pheromone trail, β —relative importance of the heuristic information;
- γ —number of ants.

Outputs:

- $NS_u(t) = \{Neigh_{u,i}(t) \mid Neigh_{u,i}(t)$ is a set of neighbors of active user u for target item i at time $t\}$, denoted as the neighborhood set of user u at time t such that each element $Neigh_{u,i}(t)$ represents the best neighbors according to the target item $i \in \tilde{I}_u(t)$.
- $BGS_u(t) = \{\text{TG\_Best}_u^c(t) \mid c \in \tilde{C}_u(t)$ and $\text{TG\_Best}_u^c(t) \subset \text{ITG}^c(t)\}$, a set of trust sub-graphs at time t such that each element $\text{TG\_Best}_u^c(t)$ represents a sub-graph consisting of the best paths in $\text{ITG}^c(t)$ that start from node u.

Computing procedure:

1. **for each** target cluster $c \in \tilde{C}_u(t)$ **do**
   /* Step 1: Initialization */
2. $G \leftarrow \text{ITG}^c(t) \in T_u(t)$ // $\text{ITG}^c(t)$ is used as graph G which models the ant colony problem
3. $Nest_G \leftarrow$ active user node (node u)
4. Assign a heuristic value to each node using the time-weighted similarity of this node to node u computed by Eq. 11
5. Generate a set $K = \{k_1, k_2, ..., k_\gamma\}$, consisting of γ artificial ants
6. Place γ ants on $Nest_G$
   /* Step 2: Solution creation */
7. Create an empty matrix $\mathbf{Solution}_{\gamma \times (\varphi+1)}$ which stores the constructed solution by each ant
8. $\mathbf{Solution}_{*,1} \leftarrow Nest_G$ // $\mathbf{Solution}_{*,1}$ represents all cells of the first column of matrix **Solution**
9. **for** l = 1, …, φ **do**
10. **for each** ant $k \in K$ located at node $a \in G$ **do**
11. **for each** node $b \in G$ **do**
12. Compute the probability of selecting node b at time t, $prob_{ab}^k(t)$, by Eq. 12
13. **end for**
14. $next\_node \leftarrow$ choose probabilistically the next node with respect to the computed probabilities
15. $\mathbf{Solution}_{k,l} \leftarrow next\_node$
16. Move ant k to the $next\_node$
17. **end for**
18. **end for**
    /* Step 3: Solution evaluation */
19. Create an empty array $\mathbf{PT}[\gamma]$ which stores the path trust for each solution
20. **for each** ant $k \in K$ **do**
21. Compute the path trust of the ant k's solution, $ptrust_k$, by Eq. 13
22. $\mathbf{PT}[\gamma] \leftarrow ptrust_k$
23. **end for**
    /* Step 4: Best solutions selection */
24. Create an empty set $Best\_paths$ which stores the best solutions
25. Pick up the maximal bpn value from array **PT** and insert corresponding bpn solutions into $Best\_paths$
26. Create a sub-graph from the best paths in G starting from node u, called $\text{TG\_Best}_u^c(t)$ (according to the selected solutions in $Best\_paths$)
27. $BGS_u(t) \leftarrow BGS_u(t) \cup \{\text{TG\_Best}_u^c(t)\}$
28. **for each** target item $i \in c$ **do**
29. **for each** node $b \in \text{TG\_Best}_u^c(t)$ **do**
30. **if** $r_{b,i} \neq null$ // $r_{b,i} \in \mathbf{R}^c(t)$ represents the user b's rating for target item $i \in c$
31. $Neigh_{u,i}(t) \leftarrow Neigh_{u,i}(t) \cup \{b\}$
32. **end if**
33. **end for**
34. $NS_u(t) \leftarrow NS_u(t) \cup \{Neigh_{u,i}(t)\}$
35. **end for**
36. **end for**
37. **return** $NS_u(t)$ and $BGS_u(t)$

ratings, STARS incorporates the temporal relevance of the ratings into the similarity computation. Using the available data in sub-matrices $\mathbf{R}^c(t)$ and $\mathbf{S}^c(t)$, STARS measures time-weighted similarities in context c. Therefore, STARS relies on both trust and similarity information to form the neighborhood of active user u in context c at time t. Actually, in each node $a \in \text{ITG}^c(t)$, the probability of choosing the next node $b \in \text{ITG}^c(t)$ depends on both "the trust value assigned by node a to node b" and "similarity between node b and the active user node in this context".

The utilization of the similarity information, as the heuristic knowledge about the nodes, along with the learned trust knowledge, memorized in the form of pheromone trails, results in some advantages:

(1) In a sparse rating matrix, the propagation of trust over the network helps to alleviate the data sparsity problem by providing extra information for neighborhood selection.
(2) The utilization of global trust values allows STARS to provide more trusted neighbors for cold users with a few or even no ratings.
(3) The utilization of the time-weighted user similarities helps STARS to adapt itself to dynamically changing user interests. Actually, as long as the active user's rating data is insufficient, she benefits from opinions of globally trusted users. As the time passes and the active user provides more information about her preferences, the probability of selecting her locally similar neighbors increases, and consequently, the pheromone value associated with the selected neighbors increases. Whenever the active user's interests change, the heuristic knowledge reflects this change and helps to select more accurate neighbors.

In the proposed ant-inspired neighborhood search algorithm, selection of the best solutions is accomplished by computing the "path trust" [53] for each constructed solution. The path trust is a function of the number of co-rated items and the trust value between two adjacent nodes in $\text{ITG}^c(t)$. The detailed steps of this algorithm are shown in Algorithm 1.

The input parameters of Algorithm 1 are as follows: the set of users U, target items $\tilde{I}_u(t)$, target clusters $\tilde{C}_u(t)$, the contextual trust data $T_u(t)$, the rating data $R_u(t)$, and the timestamp data $S_u(t)$ for active user u, the number of ants (γ), the time decay weight (θ), the maximum trust propagation distance (φ), the maximal number of the best trust paths (bpn), the relative importance of the pheromone trail (α) and heuristic value (β). As will be detailed later, parameter θ is a decay factor to emphasize users' recent preference. φ is a tunable parameter that is used to control the maximum distance from the active user to where the trust is propagated. Parameter bpn is used for identifying the best solutions (i.e., solutions with a high path trust value). Finally, α and β are two parameters that control the relative importance of trust (i.e., the pheromone value) versus similarity (i.e., the heuristic information) during neighborhood formation. Proper values of these parameters are selected by performing sensitivity analysis (see Section 4.1).

By the mentioned inputs, this algorithm finds the best neighbors of active user u at time t for each target item $i \in \tilde{I}_u(t)$. For this purpose, in each context $c \in \tilde{C}_u(t)$, it first identifies the best trust paths that start from node u in $\text{ITG}^c(t)$, and creates a sub-graph from the identified best paths, called $\text{TG\_Best}_u^c(t)$ (lines 2–27). Then, for each target item $i \in c$, those nodes of graph $\text{TG\_Best}_u^c(t)$ which have a rating for item i are selected as the neighbors of user u for item i at time t; this set of selected neighbors is represented by $Neigh_{u,i}(t)$ (lines 29–35). The mentioned whole procedure is repeated for each target cluster.

Considering a target cluster $c \in \tilde{C}_u(t)$, Algorithm 1 starts with the initialization step (lines 2–6). In this step, $\text{ITG}^c(t)$ is used as a graph which models the ant colony problem (line 2). In this graph, the weight of edgeab represents the pheromone level on this edge at time t:

$$\tau_{ab}^c(t) = W_{ab}^c(t) \tag{7}$$

where $\tau_{ab}^c(t)$ denotes the pheromone level on edge ab in $\text{ITG}^c(t)$. As mentioned in Section 3.1.1, at time $t = t_0$, for each node $a \in \text{ITG}^c(t)$, $W_{ab}^c(t)$ is equal to the global trust score of user b in the context of cluster c.

In the initialization step and after determining the underlying trust graph, the active user node, e.g., node u, is taken as the ants' nest (line 3), a heuristic value is assigned to each node (line 4), and γ artificial ants are dispatched from node u (lines 5–6). In order to determine the heuristic value of a node a, we need to compute the rating similarity between user a and active user u . In the context of cluster c, this similarity is computed using the user rating profiles available in sub-matrix $\mathbf{R}^c(t)$. To be able to adapt to changes in users' preferences, STARS incorporates the temporal information in similarity computation via weighting each rating with its temporal relevance. For this purpose, it uses a temporal relevance function which assigns a weight to each rating according to its age (time distance) with respect to the current time. In the context of cluster c, the timestamp of each observed rating until time t is available in sub-matrix $\mathbf{S}^c(t)$. At current time slot t, the temporal relevance $f_{u,i}(t)$ of the observed rating $r_{u,i}$ is computed as follows [54]:

$$f_{u,i}(t) = e^{-\theta(t-t_{u,i})} \tag{8}$$

where $\theta \in [0, 1]$ controls the decaying rate. Under the assumption that older ratings are generally less correlated with the users' current tastes and interests, the above function decreases the relevance of rating $r_{u,i}$ with the amount of time that has passed since the rating date.

To compute user similarities by incorporating time-based weights, STARS uses a modified cosine similarity measure as follows [54]:

$$TWC^c_{ua}(t) = \frac{\sum_{i \in D \cap E} (f_{u,i}(t) \cdot r_{u,i})(f_{a,i}(t) \cdot r_{a,i})}{\sqrt{\sum_{i \in D} (f_{u,i}(t) \cdot r_{u,i})^2 \sum_{i \in E} (f_{a,i}(t) \cdot r_{a,i})^2}}, \quad D = I^c_u(t), \quad E = I^c_a(t) \tag{9}$$

where $TWC^c_{ua}(t) \in [0, 1]$ represents the time-weighted cosine similarity between users u and a in context c. $TWC^c_{ua}(t)$ is computed using the available data in $\mathbf{R}^c(t)$ and $\mathbf{S}^c(t)$. The cosine similarity metric only considers the value of ratings, and does not regard the proportion of commonly rated items. So, when the number of items that have been commonly rated by two users is very small, their similarity value will be high. To solve this issue, we need to consider the number of common ratings between two users. For this purpose, we use the user-based Jaccard similarity metric [14] as a weighting factor to adjust $TWC^c_{ua}(t)$. The Jaccard metric measures the similarity based on the proportion between the number of common ratings and the total rated items up to time t, as given by:

$$Jaccard^c_{ua}(t) = \frac{|D \cap E|}{|D| + |E| - |D \cap E|}, \quad D = I^c_u(t), \quad E = I^c_a(t) \tag{10}$$

$Jaccard^c_{ua}(t)$, which represents the user-based Jaccard similarity between users u and a in context c, is computed using the available data in $\mathbf{R}^c(t)$. The combination of $TWC^c_{ua}(t)$ and $Jaccard^c_{ua}(t)$ is used to calculate the similarity $sim^c_{ua}(t)$ between users a and u in context c at time t. So, the heuristic value of node a in $\mathrm{ITG}^c(t)$, denoted as $\eta^c_a(t)$, is computed as follows:

$$\eta^c_a(t) = sim^c_{ua}(t) = Jaccard^c_{ua}(t) \times TWC^c_{ua}(t) \tag{11}$$

After the initialization step, Algorithm 1 continues with the creation of solutions (Step 2). In this step, each ant performs a depthfirst search in graph $\mathrm{ITG}^c(t)$ to select the best neighbors of the active user (lines 7–18). According to parameter φ which represents the maximum trust propagation distance, the iterative process of solution creation will continue until each ant explores a solution with depth φ. In other words, each ant creates a chain of trust relationships with the maximum size of φ. The constructed solutions are stored in a $\gamma \times (\varphi + 1)$ matrix, namely **Solution**, where each row contains the solution created by an ant (line 7). More specifically, considering an ant k, $\mathbf{Solution}_{k,*}$ (i.e., the kth row of matrix **Solution**) consists of an ordered sequence of nodes selected by ant k. The starting node of each ant (here, node u) is placed in the first cell of each row of matrix **Solution** (line 8). In order to construct a solution by an ant, STARS combines the trust information with the similarity information in a probabilistic transition rule. Considering an ant k located at node $a \in \mathrm{ITG}^c(t)$, the following probabilistic transition rule [35] is used for selecting the next movement (line 12):

$$prob^k_{ab}(t) = \begin{cases} \frac{(1+\tau^c_{ab}(t))^\alpha (1+\eta^c_b(t))^\beta}{\sum_{f \in F_k} (1+\tau^c_{af}(t))^\alpha (1+\eta^c_f(t))^\beta} & \text{if } b \in F_k \\ 0 & \text{otherwise} \end{cases} \tag{12}$$

where $prob^k_{ab}(t)$ represents the probability of selecting node $b \in \mathrm{ITG}^c(t)$, and F_k is the set of nodes that have not yet been visited by ant k. In (12), we use $1 + \eta$ instead of η for the following reason: when the active user has no rating, all the nodes have heuristic value equal to zero. If we use η in (12), then the probability of selecting any node will be equal to zero, and therefore, ants will be unable to choose their next movement. So, in this case, the system cannot provide any recommendation for the active user. Also, when the active user has a small number of ratings, the number of possible choices for the next movement will be significantly reduced, affecting the recommendation quality. Actually, in such cases, a cold user cannot benefit from opinions of globally trusted users. In order to prevent these problems, we use $1 + \eta$ instead of η in (12). For the sake of being in the same range, τ is also incremented accordingly. In this way, when the similarity between two users cannot be defined, only the trust knowledge memorized in the form of pheromone trails is considered.

After computing transition probabilities, each ant k chooses the next node to move to (line 14). To implement a probabilistic selection, we use the classical roulette-wheel procedure [60], where the nodes with higher probability have a higher chance of being the next node. When a simple linear search is used, the complexity of selection is of $\mathrm{O}(n)$, where n is the number of users. After selecting the next node (line 14), ant k stores this new node in one of the cells of the kth row of matrix **Solution** (line 15), and moves to the selected node (line 16).

The next step of Algorithm 1 is the solution evaluation (lines 19–23) in which each constructed solution is evaluated by computing the corresponding path trust [53]. The path trust of solutions constructed by ants is stored in an array of length γ, namely **PT** (line 19). As mentioned above, each constructed solution is a sequence of nodes starting from the active user node. For each constructed solution, the path trust is a function with two parameters:

1- the number of co-rated items between every two adjacent nodes x and y, and
2- the trust value issued by node x towards node y.

The paths with a high value of propagated trust can be considered as the best solutions. Actually, a chain of users with

high path trust value can provide more precise opinions for the active user. Considering solution $\mathbf{Solution}_{k,*}$ constructed by the kth ant at time t, the path trust is computed as follows (line 21):

$$ptrust_k = \frac{\sum_{\text{pair}(x,y) \text{ of adjacent elements in } \mathbf{Solution}_{k,*}} \left(\left| I_x^c(t) \cap I_y^c(t) \right| \times \tau_{xy}^c(t) \right)}{\sum_{\text{pair}(x,y) \text{ of adjacent elements in } \mathbf{Solution}_{k,*}} \left(\left| I_x^c(t) \cap I_y^c(t) \right| \right)} \tag{13}$$

$ptrust_k$ represents the path trust of the ant k's solution; x and y refer to two adjacent nodes in the constructed solution. The computed path trust of the ant k's solution is stored in $\mathbf{PT}[k]$ (line 22).

The last step of Algorithm 1 is the best solutions selection (lines 24–35). In this step, *bpn* number of solutions with the highest path trust value are added to a set, called *Best_paths*, which stores the best solutions (lines 24–25). Here, we use the median of medians algorithm [60], that is an optimal algorithm for selecting m largest elements in a list with the linear time complexity in the worst case. Afterward, Algorithm 1 must identify the best neighbors of user u for each target item belonging to the current cluster. For this purpose, it first creates a sub-graph from the best paths in $\text{ITG}^c(t)$ starting from node u, according to the selected solutions in *Best_paths* (line 26). Since this graph, called $\text{TG\_Best}_u^c(t)$, is used as a key input to our ant-inspired trust updating algorithm (Section 3.1.3), it is saved to an output set, called $BGS_u(t)$ (line 27), for later use. In the next step, for each target item $i \in c$, neighbors $Neigh_{u,i}(t)$ of user u are determined as follows: each node of graph $\text{TG\_Best}_u^c(t)$ which has a rating for item i is selected as a neighbor of user u for item i at time t (lines 29–31). All identified neighbors for different target items are stored in an output set, called $NS_u(t)$ (line 34). Finally, $NS_u(t)$ and $BGS_u(t)$ are returned as outputs (line 37).

After the execution of Algorithm 1 in the online mode, the pheromone values must be updated according to the selected neighbors. More specifically, the pheromone update process occurs after the neighborhood formation process at each time step. Using this process, at each time step, the pheromone value associated with the best neighbors of the active user in a specific context is increased, while this value for the other users is decreased. The update process, which is performed offline, is detailed in Algorithm 2 of the following section.

In the following, a simple example is given to illustrate how Algorithm 1 works.

Example 1 Suppose that there are 13 items (i_1 to i_{13}, $m = 13$) and 10 users (u_1 to u_{10}, $n = 10$). Suppose that $t_0 = 20$ (i.e., 20 days after the first rating). A sample user-item rating matrix at time $t_0 = 20$, $\mathbf{R}_{10\times13}(t_0)$, is depicted in Table 1. The ratings are integers ranged from 1 to 5. We use symbol? to state that a user has not rated an item yet. A sample timestamp matrix of the observed ratings, $\mathbf{S}_{10\times13}(t_0)$, is also depicted in Table 2. Time is measured with the day granularity, and timestamps are ranged from 1 (the first known rating time) to 20 (the current time). We assume that item clusters obtained based on the semantic similarities are $c_1 = \{i_1, i_2\}$, $c_2 = \{i_3, i_4, i_5, i_6, i_7\}$, $c_3 = \{i_8, i_9, i_{10}, i_{11}\}$ and $c_4 = \{i_{12}, i_{13}\}$. So, there are four rating sub-matrices at time $t = t_0$: $\mathbf{R}_{10\times2}^{c_1}(t_0)$, $\mathbf{R}_{10\times5}^{c_2}(t_0)$, $\mathbf{R}_{10\times4}^{c_3}(t_0)$ and $\mathbf{R}_{10\times2}^{c_4}(t_0)$. Also, we have four timestamp sub-matrices at time $t = t_0$: $\mathbf{S}_{10\times2}^{c_1}(t_0)$, $\mathbf{S}_{10\times5}^{c_2}(t_0)$, $\mathbf{S}_{10\times4}^{c_3}(t_0)$ and $\mathbf{S}_{10\times2}^{c_4}(t_0)$.

Based on the available ratings in a rating sub-matrix, STARS computes the global trust score of each user in the related context using (5). As shown in Table 3, users u_4, u_7, u_8 and u_5 are the most globally trusted users in the context of clusters c_1, c_2, c_3 and c_4, respectively. After that, STARS creates four initial ITGs, namely $\text{ITG}^{c_1}(t_0)$, $\text{ITG}^{c_2}(t_0)$, $\text{ITG}^{c_3}(t_0)$ and $\text{ITG}^{c_4}(t_0)$. Each ITG is a fully connected directed graph that consists of 10 nodes and 90 (i.e., $10 \times (10 - 1)$) edges. Based on the calculated trust scores (Table 3), the initial weight of each edge in an ITG is determined using (6). As mentioned before, in the initial ITGs, the estimated trust in a certain user is the same for every user. As the time passes and users provide more information about their preferences, STARS locally adjusts the weight of corresponding links in the ITGs.

Now, we assume that user u_{10} is the active user. Based on Table 1, target items and target clusters for this user at $t = t_0$ are $\tilde{I}_{u_{10}}(t_0) = \{i_1, i_2, i_3, i_6, i_8\}$and $\tilde{C}_{u_{10}}(t_0) = \{c_1, c_2, c_3\}$, respectively. In the context of cluster c_1, user u_{10} is an extreme cold user since she has not provided any rating. The context-related data for this user at time $t = t_0$ consists of $T_{u_{10}}(t_0) = \{\text{ITG}^{c_1}(t_0), \text{ITG}^{c_2}(t_0), \text{ITG}^{c_3}(t_0)\}$, $R_{u_{10}}(t_0) = \{\mathbf{R}^{c_1}(t_0), \mathbf{R}^{c_2}(t_0), \mathbf{R}^{c_3}(t_0)\}$, and $S_{u_{10}}(t_0) = \{\mathbf{S}^{c_1}(t_0), \mathbf{S}^{c_2}(t_0), \mathbf{S}^{c_3}(t_0)\}$. We apply Algorithm 1 in order to determine the best neighbors of user u_{10} for each target item at time $t = t_0$.

Let $\gamma = 10, \theta = 0.3, \varphi = 2, bpn = 6, \alpha = 2$, and $\beta = 1$. In the initialization step (lines 2–6), Algorithm 1 determines the underlying trust graph to apply the ACO algorithm, and initializes heuristic values using (11). The result is shown

Table 1 The user–item rating matrix at time $t = t_0$

User	Cluster												
	c_1			c_2				c_3				c_4	
	Item												
	i_1	i_2	i_3	i_4	i_5	i_6	i_7	i_8	i_9	i_{10}	i_{11}	i_{12}	i_{13}
u_1	?	?	4	3	5	?	3	1	?	4	5	2	?
u_2	3	5	3	?	4	?	1	?	?	5	3	?	?
u_3	?	3	?	?	2	1	3	2	4	?	2	?	3
u_4	1	2	2	?	3	4	?	3	4	5	?	?	?
u_5	?	?	1	?	?	4	?	?	3	1	?	4	2
u_6	?	2	?	2	5	3	?	2	5	?	?	?	1
u_7	3	?	2	?	4	?	3	?	3	?	1	3	4
u_8	1	4	?	3	?	3	?	?	2	2	3	?	?
u_9	?	?	1	?	2	5	?	3	?	4	3	?	?
u_{10}	?	?	?	3	5	?	2	?	2	5	3	4	1

Table 2 The timestamp matrix at time $t = t_0$

User	Cluster												
	c_1			c_2				c_3				c_4	
	Item												
	i_1	i_2	i_3	i_4	i_5	i_6	i_7	i_8	i_9	i_{10}	i_{11}	i_{12}	i_{13}
u_1	–	–	17	17	2	–	2	11	–	11	6	19	–
u_2	19	15	2	–	1	–	1	–	–	3	7	–	–
u_3	–	14	–	–	20	20	20	9	13	–	13	–	10
u_4	15	11	14	–	20	6	–	10	10	8	–	–	–
u_5	–	–	4	–	–	12	–	–	2	19	–	10	7
u_6	–	14	–	18	3	1	–	20	1	–	–	–	13
u_7	11	–	10	–	2	–	2	–	4	–	15	7	4
u_8	19	3	–	5	–	7	–	–	11	15	11	–	–
u_9	–	–	4	–	18	4	–	3	–	3	8	–	–
u_{10}	–	–	–	19	1	–	1	–	18	2	8	1	19

Table 3 The initial global trust score of each user according to different item clusters

User	$Gtrust_v^{c_1}(t_0)$	$Gtrust_v^{c_2}(t_0)$	$Gtrust_v^{c_3}(t_0)$	$Gtrust_v^{c_4}(t_0)$
u_1	0	0.4510	0.4374	0.4767
u_2	0.6840	0.4860	0.4503	0
u_3	0.6075	0.3791	0.4553	0.6075
u_4	0.6870	0.4762	0.4599	0
u_5	0	0.3330	0.3748	0.6680
u_6	0.6075	0.3818	0.3520	0.6075
u_7	0.4767	0.5032	0.3910	0.6470
u_8	0.6630	0.2950	0.5423	0
u_9	0	0.4402	0.4688	0
u_{10}	0	0.4012	0.5197	0.6390

Table 4 Similarity between active user u_{10} and another user a in different contexts

User	$sim^{c_1}_{u_{10}a}(t_0)$	$sim^{c_2}_{u_{10}a}(t_0)$	$sim^{c_3}_{u_{10}a}(t_0)$
u_1	0	0.7499	0.2565
u_2	0	0.4954	0.6539
u_3	0	0.4120	0.4625
u_4	0	0.2000	0.4182
u_5	0	0	0.0256
u_6	0	0.4999	0.2500
u_7	0	0.4828	0.1215
u_8	0	0.2500	0.3103
u_9	0	0.2000	0. 4999

Table 5 The constructed probabilistic solutions for active user u_{10} in each trust graph

Ant	Trust graph								
	$ITG^{c_1}(t_0)$			$ITG^{c_2}(t_0)$			$ITG^{c_3}(t_0)$		
	Solution$_{10\times 3}$			**Solution**$_{10\times 3}$			**Solution**$_{10\times 3}$		
k_1	u_{10}	u_3	u_6	u_{10}	u_7	u_2	u_{10}	u_8	u_9
k_2	u_{10}	u_8	u_3	u_{10}	u_6	u_1	u_{10}	u_4	u_9
k_3	u_{10}	u_4	u_6	u_{10}	u_2	u_9	u_{10}	u_9	u_3
k_4	u_{10}	u_2	u_7	u_{10}	u_7	u_6	u_{10}	u_2	u_4
k_5	u_{10}	u_8	u_2	u_{10}	u_2	u_6	u_{10}	u_8	u_3
k_6	u_{10}	u_8	u_2	u_{10}	u_1	u_6	u_{10}	u_2	u_4
k_7	u_{10}	u_4	u_3	u_{10}	u_2	u_4	u_{10}	u_4	u_3
k_8	u_{10}	u_2	u_4	u_{10}	u_1	u_9	u_{10}	u_8	u_4
k_9	u_{10}	u_3	u_4	u_{10}	u_1	u_4	u_{10}	u_9	u_8
k_{10}	u_{10}	u_4	u_6	u_{10}	u_2	u_1	u_{10}	u_2	u_9

Table 6 The path trusts of constructed solutions for active user u_{10} in each trust graph

Ant	Trust graph		
	$ITG^{c_1}(t_0)$ **PT**$[k]$	$ITG^{c_2}(t_0)$ **PT**$[k]$	$ITG^{c_3}(t_0)$ **PT**$[k]$
k_1	0.6075	0.4929	0.5129
k_2	0.6075	0.4164	0.4643
k_3	0.6075	0.4631	0.4620
k_4	0.4767	0.4627	0.4535
k_5	0.6840	0.4513	0.5075
k_6	0.6840	0.4233	0.4535
k_7	0.6075	0.4811	0.4576
k_8	0.6870	0.4466	0.5093
k_9	0.6870	0.4611	0.5055
k_{10}	0.6075	0.4650	0.4595

Table 7 The best trust paths starting from node u_{10} in each trust graph

Trust graph	*Best_paths*
$ITG^{c_1}(t_0)$	{**Solution**$_{1,*}$, **Solution**$_{2,*}$, **Solution**$_{5,*}$, **Solution**$_{6,*}$, **Solution**$_{8,*}$, **Solution**$_{9,*}$}
$ITG^{c_2}(t_0)$	{**Solution**$_{1,*}$, **Solution**$_{3,*}$, **Solution**$_{4,*}$, **Solution**$_{7,*}$, **Solution**$_{9,*}$, **Solution**$_{10,*}$}
$ITG^{c_3}(t_0)$	{**Solution**$_{1,*}$, **Solution**$_{2,*}$, **Solution**$_{3,*}$, **Solution**$_{5,*}$, **Solution**$_{8,*}$, **Solution**$_{9,*}$}

in Table 4 as the similarity between u_{10} and other users according to different target clusters.

In Step 2 of Algorithm 1, each ant selects its next hop using (12) and constructs a trust path of length $\varphi = 2$ (lines 7–18). Table 5 shows the probabilistic solutions constructed by ants in each trust graph. To evaluate each solution (lines 19–23), Algorithm 1 computes its path trust using (13); the results are given in Table 6. As an example related to graph $ITG^{c_2}(t_0)$, consider the solution **Solution**$_{2,*} = [u_{10}, u_6, u_1]$ constructed by the second ant. Recall that the kth row of matrix **Solution** contains the solution constructed by ant k. The path trust of **Solution**$_{2,*}$ is computed as follows: **PT**[2] $= (0.3818 \times 2 + 0.4510 \times 2)/(2+2) = 0.4164$. In the next step, $bpn = 6$ number of solutions with the highest path trust value are added to set *Best_paths* (line 25), as shown in Table 7.

For instance, the results presented in Table 6 show that in the context of cluster c_2, the solutions constructed by ants 1, 3, 4, 7, 9 and 10 have the highest path trust value, so they are selected as the best solutions in this context . According to the selected solutions in *Best_paths*, Algorithm 1 creates the sub-graphs consisting of the best trust paths that start from node u_{10} in each trust graph (line 26). For example, TG_Best $^{c_2}_{u_{10}}(t_0)$ contains the best trust paths constructed by the ants 1, 3, 4, 7, 9 and 10. The resulting graphs from this step are shown in Fig. 2.

Finally, based on the selected paths, the best neighbors of active user u_{10} for different target items at time $t = t_0$ are determined; these neighbors are given in Table 8. For example, consider target item $i_1 \in c_1$. Among the nodes of graph TG_Best$^{c_1}_{u_{10}}(t_0)$, nodes u_2, u_4 and u_8 have a rating for item i_1 (refer to Table 1), and thus, they are selected as the neighbors of user u_{10} for item i_1 at time $t = t_0$.

In this example, the outputs of Algorithm 1 are as follows:

1) $NS_{u_{10}}(t_0) = \{Neigh_{u_{10},i}(t_0) | i \in \tilde{I}_{u_{10}}(t_0)\}$

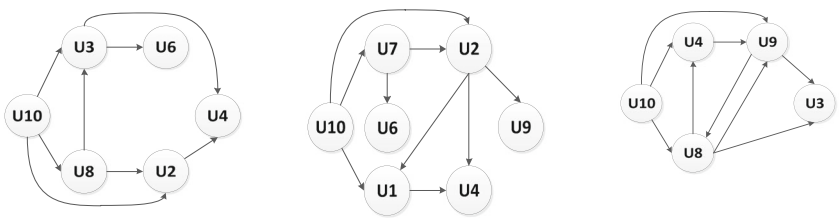
2) $BGS_{u_{10}}(t_0) = \{\text{TG\_Best}^c_{u_{10}}(t_0) | c \in \tilde{C}_{u_{10}}(t_0)\}$

Generating recommendations After neighborhood formation, STARS computes the ratings of all items that have not been rated by an active user u until time t. The predicted rating of active user u on target item $i \in \tilde{I}_u(t)$ is calculated using the weighted average of deviations from the neighbor's mean rating. Assume that target item i belongs to cluster c. Based on the available ratings in $\mathbf{R}^c(t)$, the predicted rating of active user u for target item i, $\hat{r}_{u,i}$, is computed as follows:

$$\hat{r}_{u,i} = \overline{r_u} + \frac{\sum_{v \in Neigh_{u,i}(t)} (r_{v,i} - \overline{r_v}) \times IW^c_{uv}(t)}{\sum_{v \in Neigh_{u,i}(t)} IW^c_{uv}(t)} \tag{14}$$

where $\overline{r_u}$ and $\overline{r_v}$ are the mean ratings of active user u and neighbor v for items belonging to cluster c, respectively; $IW^c_{uv}(t)$ denotes the importance weight of the user v's ratings relative to the user u's ratings in context c at time t. The importance weight consists of two parts: trust value $\tau^c_{uv}(t)$ that node u holds about node v in the context of cluster c at time t, and similarity $sim^c_{uv}(t)$ between users u and v in context c at time t. $IW^c_{uv}(t)$ is computed using the harmonic mean to combine both trust and similarity values as follows:

$$IW^c_{uv}(t) = \frac{2 \times (1 + \tau^c_{uv}(t)) \times (1 + sim^c_{uv}(t))}{\tau^c_{uv}(t) + sim^c_{uv}(t) + 2} \tag{15}$$



(a) TG_Best $^{c_1}_{u_{10}}(t_0)$ (b) TG_Best $^{c_2}_{u_{10}}(t_0)$ (c) TG_Best $^{c_3}_{u_{10}}(t_0)$

Fig. 2 Sub-graphs consisting of the best trust paths that start from node u_{10} in each context-related trust graph

Table 8 The best neighbors of user u_{10} for each target item $i \in \tilde{I}_{u_{10}}$ at time $t = t_0$

Target items	$Neigh_{u_{10},i}(t_0)$
i_1	$\{u2, u4, u8\}$
i_2	$\{u2, u3, u4, u6, u8\}$
i_3	$\{u1, u2, u4, u7, u9\}$
i_6	$\{u4, u6, u9\}$
i_8	$\{u3, u4, u9\}$

When an active user has no rating, the similarity between her and all other users is equal to zero. Therefore, we use $1 + sim$ instead of sim in (15), and thus, τ is also incremented accordingly. The harmonic mean is robust to large differences between inputs so that high values can only be obtained when both similarity and trust values are high.

Finally, STARS ranks the predicted ratings and recommends those N items that have the highest predicted rating.

3.1.3 Trust network updater module

The aim of this module is to update each trust graph $\text{ITG}^c(t) \in T_u(t)$ such that the pheromone value associated with the best neighbors of user u in context c is increased, while this value for the other users is decreased through pheromone evaporation.

For this purpose, STARS uses a novel ant-inspired trust updating algorithm. In this algorithm, the amount of pheromone deposited on the edge connecting active user u and a trustworthy neighbor v depends on: (1) the inferred trust value from user u to user v through the trust paths selected in Algorithm 1, and (2) the amount of confidence in user u's opinion about user v. Confidence is directly related to the number of co-rated items between two users. The detailed steps of this algorithm are shown in Algorithm 2.

In Algorithm 2, the set of users U, the set of target items $\tilde{I}_u(t)$, the corresponding target clusters $\tilde{C}_u(t)$, the contextual trust data $T_u(t)$ and the rating data $R_u(t)$ for active user u, the maximum trust propagation distance (φ), the pheromone decay value (ρ), as well as the outputs of Algorithm 1 are taken as inputs. Parameter ρ regulates the pheromone evaporation between times t and $t+1$ to avoid unlimited accumulation of the pheromone. The goal of this algorithm is to update each trust graph $\text{ITG}^c(t) \in T_u(t)$ and store the corresponding updated graph, $\text{ITG}^c(t+1)$, in the database. By considering $\text{ITG}^c(t)$, during lines 2–8, this algorithm decreases trust intensity by a small constant ρ on all the edges connecting active user u and other users (refer to (16) and (21)). In addition, trust intensity is increased

Algorithm 2 Ant-inspired trust updating for active user u at time t

Inputs:

- $U = \{u_1, u_2, ..., u_n\}$, $n > 1$—the user set;
- $\tilde{I}_u(t)$—the set of target items for user u at time t;
- $\tilde{C}_u(t)$—the corresponding target clusters of $\tilde{I}_u(t)$;
- $T_u(t)$—the set of implicit trust graphs at time t, according to the target clusters $\tilde{C}_u(t)$;
- $R_u(t)$—the set of rating sub-matrices at time t, according to the target clusters $\tilde{C}_u(t)$;
- $NS_u(t)$—the neighborhood set of user u at time t;
- $BGS_u(t)$—the set of the best trust paths starting from node u in each $\text{ITG}^c(t) \in T_u(t)$;
- φ—the maximum trust propagation distance;
- ρ—pheromone decay value.

Outputs:

- Each graph $\text{ITG}^c(t) \in T_u(t)$ is updated, and the corresponding updated graph, $\text{ITG}^c(t+1)$, is stored in the database.

Computing procedure:

1. **for each** target cluster $c \in \tilde{C}_u(t)$ **do**
2. **for each** user $v \in U$ **do**
3. **if** there exists an item $i \in c$ such that $v \in Neigh_{u,i}(t)$
4. Update the pheromone deposited on edge uv of $\text{ITG}^c(t)$ using (16) // pheromone evaporation and deposition
5. **else**
6. Update the pheromone deposited on edge uv of $\text{ITG}^c(t)$ using (21) // pheromone evaporation
7. **end if**
8. **end for**
9. **end for**

(refer to (16)) by a small quantity $\Delta\tau^c(t)$ on the edges connecting active user u and users who have been selected as her best neighbors for at least one item $i \in c$.

Considering a target cluster c, for each user v in the user set U, if user v is the best neighbor of active user u for at least one item $i \in c$ (line 3), then the pheromone laid on edge uv of $\text{ITG}^c(t)$ is updated as follows (line 4):

$$\tau^c_{uv}(t+1) = W^c_{uv}(t+1) = (1-\rho) \times \tau^c_{uv}(t) + \Delta\tau^c_{uv}(t) \quad (16)$$

where $\tau^c_{uv}(t+1)$ represents the pheromone level on edge uv at time $t+1$ which is equal to weight $W^c_{uv}(t+1)$ of this edge

at time $t+1$; similarly, $\tau^c_{uv}(t)$ represents the pheromone level at time t. $\Delta\tau^c_{uv}(t)$ is the amount of pheromone deposited on edge uv of ITG$^c(t)$ at time t. It is computed using sub-graph TG_Best$^c_u(t) \in BGS_u(t)$ which contains the best trust paths in ITG$^c(t)$ that start from node u. In order to compute $\Delta\tau^c_{uv}(t)$, Algorithm 2 should calculate the following parameters using TG_Best$^c_u(t)$:

1) $Inf\_Trust^c_{uv}(t)$: the inferred trust value between users u and v through the existing trust paths in TG_Best$^c_u(t)$. Let $P = \{p_1, p_2, ..., p_\psi\}$, $\psi \geq 1$, be a set of paths between two nodes u and v in graph TG_Best$^c_u(t)$, and $p_j = [a_0, a_1, ..., a_{q-1}, a_q]$, $p_j \in P$, is a path of length $1 \leq q \leq \varphi$ from $a_0 = u$ to $a_q = v$. We note that for each path $p_j \in P$, there exists an ant $k \in K$ such that p_j is a sub-path of solution $\mathbf{Solution}_{k,*}$ traversed by the kth ant. Then, $Inf\_Trust^c_{uv}(t)$ is recursively computed as follows:

$$Inf\_Trust^c_{a_0a_q}(t) = \begin{cases} \tau^c_{a_0a_q}(t) & \text{if there exists a path } p_j \in P \text{ such that } a_0 \text{ is adjacent to } a_q \\ \dfrac{\sum_{p_j \in P} Inf\_Trust^c_{a_0a_{q-1}}(t) \times \left(\tau^c_{a_{q-1}a_q}(t) \times \delta\right)}{\sum_{p_j \in P} \tau^c_{a_0a_1}(t)}, \delta = \dfrac{\varphi - d_{a_0a_q} + 1}{\varphi} & \text{otherwise} \end{cases} \tag{17}$$

where δ is a weighting parameter, and $d_{a_0a_q} \in [2, \varphi]$ represents the trust propagation distance from source node a_0 to destination node a_q. The value of $Inf\_Trust^c_{uv}(t)$ varies within the range (0, 1]. It does not take zero value since the probability of selecting edges with no pheromone is equal to zero (see (12)). As shown in (17), when there is a direct trust link between users u and v in TG_Best$^c_u(t)$, the value of $Inf\_Trust^c_{uv}(t)$ is equal to the weight (or pheromone) of that link. Otherwise, trust propagation is needed to compute the indirect trust value between users u and v. Given path $p_j = [a_0, a_1, ..., a_{q-1}, a_q]$, the indirect trust value between two nodes a_0 and a_q depends on: (1) the direct trust value between two adjacent nodes a_{q-1} and a_q; and (2) the propagated trust value between nodes a_0 and a_{q-1}, which is calculated recursively through intermediate nodes a_1 to a_{q-2}. It is possible that there are multiple paths between two users. In such cases, a trust aggregation method is needed to combine the different trust beliefs that user u has received about v. To compute the propagated implicit trust between users u and v based on the available trust paths in P, Algorithm 2 uses a weighted mean aggregation method due to its robust performance [14]. We adopt the method presented by Shambour and Lu [14] which estimates a trust value for a particular user on distance $d > 1$ based on the estimated trust values for users on distance $d-1$. The weighting parameter δ is used in order to ensure that trust decreases along the propagation. δ will ensure that trust scores from the closer trusted neighbors will have more influence on the trust propagation process.

2) $Conf^c_{uv}(t)$: the amount of confidence in the user u's opinion about user v in the context of cluster c at time t. This parameter expresses the reliability of the association between users u and v in TG_Best$^c_u(t)$. To compute $Conf^c_{uv}(t)$, we adopt the model proposed by Papagelis et al. [26]. In this model, confidence is directly related to the number of co-rated items between two users. In other words, the more items two users have co-rated, the higher the degree of confidence their association would have. In order to compute the confidence of all direct associations of a user, this model first identifies the most confident direct association for this user. Then, confidence values of the remaining direct associations are calculated with respect to the identified most confident association.

At time t, STARS computes the confidence between two users in the context of cluster c according to the available ratings in $\mathbf{R}^c(t)$. Based on the existing trust paths in P (recall that P is the set of paths between two nodes u and v in graph TG_Best$^c_u(t)$), $Conf^c_{uv}(t) \in [0, 1]$ is computed as follows:

$$Conf^c_{a_0a_q}(t) = \begin{cases} \dfrac{\left|I^c_{a_0}(t) \cap I^c_{a_q}(t)\right|}{\left|I^c_{a_0}(t) \cap I^c_{Max\_Conf\_Node}(t)\right|} & \text{if there exists a path } p_j \in P \text{ such that } a_0 \text{ is adjacent to } a_q \\ \dfrac{\sum_{p_j \in P} Conf^c_{a_0a_1}(t) \times Conf^c_{a_1a_q}(t)}{|P|} & \text{otherwise} \end{cases} \tag{18}$$

where *Max_Conf_Node* denotes the most confident association in the trust network of source user a_0. Actually, among the direct associations of source node a_0 in graph TG_Best$^c_u(t)$, *Max_Conf_Node* is a node that

Table 9 The best trust paths between user u_{10} and her neighbors according to graph $\text{TG\_Best}^{c_2}_{u_{10}}(t_0)$

Neighbor	The best trust paths from node u_{10} to a neighbor node v
u_1	$P = \{p_1 = [u_{10}, u_1],\ p_2 = [u_{10}, u_2, u_1]\}$
u_2	$P = \{p_1 = [u_{10}, u_2],\ p_2 = [u_{10}, u_7, u_2]\}$
u_4	$P = \{p_1 = [u_{10}, u_2, u_4],\ p_2 = [u_{10}, u_1, u_4]\}$
u_6	$P = \{p_1 = [u_{10}, u_7, u_6]\}$
u_7	$P = \{p_1 = [u_{10}, u_7]\}$
u_9	$P = \{p_1 = [u_{10}, u_2, u_9]\}$

has the largest number of co-rated items with a_0. As shown in (18), when there is a direct trust link between two users in $\text{TG\_Best}^c_u(t)$, the confidence value is directly calculated based on the number of co-rated items between two users. Otherwise, it is calculated through a set of intermediate nodes [26].

After computing $Inf\_Trust^c_{uv}(t)$ and $Conf^c_{uv}(t)$, $\Delta\tau^c_{uv}(t)$ is calculated using the harmonic mean to integrate these two parameters. This is because high values of $\Delta\tau^c_{uv}(t)$ can only be obtained when both confidence and trust values are high. The following formula is used to compute $\Delta\tau^c_{uv}(t) \in (0, 1]$:

$$\Delta\tau^c_{uv}(t) = \frac{1}{2} H\left([1+Inf\_trust^c_{uv}(t)],[1+Conf^c_{uv}(t)]\right) \times \lambda^c_{uv}(t) \tag{19}$$

where H denotes the harmonic mean of the parameters. In addition to the confidence and trust parameters, $\Delta\tau^c_{uv}(t)$ has a direct relationship with parameter $\lambda^c_{uv}(t)$ representing the number of times user v has been selected as a neighbor of user u in context c at time t. It allows to deposit more pheromone on the edges leading to those neighbors who are selected more frequently. According to the available ratings in $\mathbf{R}^c(t)$, $\lambda^c_{uv}(t)$ is calculated as follows:

$$\lambda^c_{uv}(t) = \frac{\left|I^c_v(t) \cap I'^c_u(t)\right|}{\left|I'^c_u(t)\right|} \tag{20}$$

where $I'^c_u(t)$ refers to the set of items in cluster c that have not been rated by user u until time t.

As shown in (19), $Conf^c_{uv}(t)$ and $Inf\_Trust^c_{uv}(t)$ are incremented by one. The reason is that $\Delta\tau^c_{uv}(t)$ must be greater than zero for the best neighbors, whereas the value of $Conf^c_{uv}(t)$ may be equal to zero. Since STARS is initialized with global trust values, it is possible that user u trusts user v even if they have no co-rated items. In this case, if we use $Conf^c_{uv}(t)$ in (19), then $\Delta\tau^c_{uv}(t)$ will be equal to zero, and therefore, only pheromone decay will occur on edge uv. To prevent this, $Conf^c_{uv}(t)$ is incremented by one. For the sake of being in the same range, $Inf\_Trust^c_{uv}(t)$ is also incremented accordingly. This allows more pheromone to be deposited for trusted neighbors who can be recognized as more confident than others. By incrementing the arguments of H, the maximum value of $\Delta\tau^c_{uv}(t)$ will be equal to 2. This may cause the algorithm to cover the initial selected solutions and ignore others. To prevent this, we divide the harmonic mean by 2. Therefore, $\Delta\tau^c_{uv}(t)$ is a small quantity whose maximum value is equal to 1.

As mentioned before, if user v has appeared at least once as a neighbor of active user u, then the corresponding pheromone value will be updated using (16); otherwise, it will be updated using (21). According to (21), only pheromone decay occurs for such users:

$$\tau^c_{uv}(t+1) = W^c_{uv}(t+1) = (1-\rho) \times \tau^c_{uv}(t) \tag{21}$$

In the following, a simple example illustrates how Algorithm 2 can be used to update trust graphs.

Example 2 From our previous example, the output sets $NS_{u_{10}}(t_0)$ and $BGS_{u_{10}}(t_0)$ were obtained. Now, we apply Algorithm 2 in order to update the weight of each outgoing link from node u_{10} in $\text{ITG}^c(t_0)$, where $c \in \tilde{C}_{u_{10}}(t_0)$. Let

Table 10 The inferred trust value between user u_{10} and her best neighbors in the context of cluster c_2 at time $t = t_0$

Neighbor	$Inf\_Trust^{c_2}_{u_{10}v}(t_0)$
u_1	$\tau^{c_2}_{u_{10}u_1}(t_0) = 0.4510$
u_2	$\tau^{c_2}_{u_{10}u_2}(t_0) = 0.4860$
u_4	$\left(\left(\tau^{c_2}_{u_{10}u_2}(t_0) \times \tau^{c_2}_{u_2u_4}(t_0) \times \delta\right) + \left(\tau^{c_2}_{u_{10}u_1}(t_0) \times \tau^{c_2}_{u_1u_4}(t_0) \times \delta\right)\right) / \left(\tau^{c_2}_{u_{10}u_2}(t_0) + \tau^{c_2}_{u_{10}u_1}(t_0)\right) = 0.2381$
u_6	$\left(\tau^{c_2}_{u_{10}u_7}(t_0) \times \tau^{c_2}_{u_7u_6}(t_0) \times \delta\right) / \tau^{c_2}_{u_{10}u_7}(t_0) = 0.1909$
u_7	$\tau^{c_2}_{u_{10}u_7}(t_0) = 0.5032$
u_9	$\left(\tau^{c_2}_{u_{10}u_2}(t_0) \times \tau^{c_2}_{u_2u_9}(t_0) \times \delta\right) / \tau^{c_2}_{u_{10}u_2}(t_0) = 0.2201$

Table 11 The confidence value between user u_{10} and her best neighbors in the context of cluster c_2 at time $t = t_0$

Neighbor	$Conf^{c_2}_{u_{10}v}(t_0)$
u_1	$\left\|I^{c_2}_{u_{10}}(t_0) \cap I^{c_2}_{u_1}(t_0)\right\| / \left\|I^{c_2}_{u_{10}}(t_0) \cap I^{c_2}_{u_1}(t_0)\right\| = 3/3 = 1$
u_2	$\left\|I^{c_2}_{u_{10}}(t_0) \cap I^{c_2}_{u_2}(t_0)\right\| / \left\|I^{c_2}_{u_{10}}(t_0) \cap I^{c_2}_{u_1}(t_0)\right\| = 2/3 = 0.6667$
u_4	$\frac{\left(\frac{\left\|I^{c_2}_{u_{10}}(t_0)\cap I^{c_2}_{u_1}(t_0)\right\|}{\left\|I^{c_2}_{u_{10}}(t_0)\cap I^{c_2}_{u_1}(t_0)\right\|} \times \frac{\left\|I^{c_2}_{u_1}(t_0)\cap I^{c_2}_{u_4}(t_0)\right\|}{\left\|I^{c_2}_{u_1}(t_0)\cap I^{c_2}_{u_4}(t_0)\right\|}\right) + \left(\frac{\left\|I^{c_2}_{u_{10}}(t_0)\cap I^{c_2}_{u_2}(t_0)\right\|}{\left\|I^{c_2}_{u_{10}}(t_0)\cap I^{c_2}_{u_1}(t_0)\right\|} \times \frac{\left\|I^{c_2}_{u_2}(t_0)\cap I^{c_2}_{u_4}(t_0)\right\|}{\left\|I^{c_2}_{u_2}(t_0)\cap I^{c_2}_{u_1}(t_0)\right\|}\right)}{2} = 0.7222$
u_6	$\left(\left\|I^{c_2}_{u_{10}}(t_0) \cap I^{c_2}_{u_7}(t_0)\right\| / \left\|I^{c_2}_{u_{10}}(t_0) \cap I^{c_2}_{u_1}(t_0)\right\|\right) \times \left(\left\|I^{c_2}_{u_7}(t_0) \cap I^{c_2}_{u_6}(t_0)\right\| / \left\|I^{c_2}_{u_7}(t_0) \cap I^{c_2}_{u_2}(t_0)\right\|\right) = 0.2222$
u_7	$\left\|I^{c_2}_{u_{10}}(t_0) \cap I^{c_2}_{u_7}(t_0)\right\| / \left\|I^{c_2}_{u_{10}}(t_0) \cap I^{c_2}_{u_1}(t_0)\right\| = 2/3 = 0.6667$
u_9	$\left(\left\|I^{c_2}_{u_{10}}(t_0) \cap I^{c_2}_{u_2}(t_0)\right\| / \left\|I^{c_2}_{u_{10}}(t_0) \cap I^{c_2}_{u_1}(t_0)\right\|\right) \times \left(\left\|I^{c_2}_{u_2}(t_0) \cap I^{c_2}_{u_9}(t_0)\right\| / \left\|I^{c_2}_{u_2}(t_0) \cap I^{c_2}_{u_1}(t_0)\right\|\right) = 0.4444$

us consider the updating process for the trust graph corresponding to target cluster c_2. According to $NS_{u_{10}}(t_0)$, users u_1, u_2, u_4, u_6, u_7 and u_9 have been selected as the best neighbors of user u_{10} with respect to those items belonging to cluster c_2 (refer to Table 8). Table 9 shows the best trust paths between user u_{10} and her neighbors according to graph $\text{TG\_Best}^{c_2}_{u_{10}}(t_0)$. According to these paths, Algorithm 2 applies (16) to update the weight of each outgoing link from node u_{10} to a neighbor node v in $\text{ITG}^{c_2}(t_0)$ (line 4). For this purpose, it computes the inferred trust (refer to (17)) and confidence value (refer to (18)) between user u_{10} and the mentioned neighbors, as shown in Tables 10 and 11, respectively. Suppose that the evaporation rate ρ is equal to 0.1. On the basis of Tables 10 and 11, Algorithm 2 applies (19) to calculate $\Delta\tau^{c_2}_{u_{10}v}(t_0)$; the results are given in Table 12. Now, using (16), the pheromone value associated with the selected neighbors is increased. For those users who have not been selected as a neighbor, Algorithm 2 updates their corresponding pheromone value using (21). At last, the obtained $\text{ITG}^{c_2}(t_0 + 1)$ (refer to Table 13) is stored in the database for later use.

3.2 Computational complexity analysis

In this section, we analyze the computational complexity of the online phase of STARS. The preprocessor and the trust network updater modules work in the offline mode, while the recommender module is the online component of STARS. The computational complexity of the recommender module, which is responsible for generating a list of top-N recommendations for users, is the combination of the computational complexities of its components. Considering an active user, first, $O(m)$ is required to retrieve the context-specific data for m items. Then, as will be detailed in the next paragraph, $O(mn + 2z\varphi\gamma n + m\gamma\varphi)$ is the upper bound on the complexity required to select the best neighborhood using Algorithm 1. Finally, the time required to produce a list of recommendations for the active user is of $O(mN)$ because we need to predict and rank all unrated items and recommends those N items that have the highest predicted rating. We note that for a typical top-N recommender system, N is a small value and usually takes values between 10 and 50. Therefore, the overall computational complexity of this module becomes $O(m) + O(mn + 2z\varphi\gamma n + m\gamma\varphi) + O(mN) \approx O(mn + 2z\varphi\gamma n + m\gamma\varphi)$. So, in the online phase, the computational effort is mainly spent by Algorithm 1 for selecting the best neighborhood. In the following, the computational complexity analysis of this algorithm is detailed.

Considering an active user, the upper bound complexity of Algorithm 1 is divided into four steps: first, $O(mn)$ is required for the initialization step. In this step, the amount of time required to compute the similarity between the active user and other users in a specific context c is $O(|c|\, n)$. Thus, $O(mn)$ is required for computing the similarities between the profile of the active user and profiles of other users in different contexts. In the second step, the main computation is the process of constructing solutions in different contexts

Table 12 Computation of $\Delta\tau^{c_2}_{u_{10}v}(t_0)$

User	H	$\lambda^{c_2}_{u_{10}v}(t_0)$	$\Delta\tau^{c_2}_{u_{10}v}(t_0)$
u_1	1.6818	0.5	0.4205
u_2	1.5712	0.5	0.3928
u_4	1.4406	1	0.7203
u_6	1.2063	0.5	0.3016
u_7	1.5807	0.5	0.3952
u_9	1.3228	1	0.6614

Table 13 Pheromone on the edges linking user u_{10} with other users on $\mathrm{ITG}^{c_2}(t_0+1)$

User	$\tau^{c_2}_{u_{10}v}(t_0)$	Evaporation	Deposition	$\tau^{c_2}_{u_{10}v}(t_0+1)$
u_1	0.4510	0.0451	0.4205	0.8264
u_2	0.4860	0.0486	0.3928	0.8302
u_3	0.3791	0.0379	0	0.3412
u_4	0.4762	0.0476	0.7203	1.1489
u_5	0.3330	0.0333	0	0.2997
u_6	0.3818	0.0382	0.3016	0.6452
u_7	0.5032	0.0503	0.3952	0.8481
u_8	0.2950	0.0295	0	0.2655
u_9	0.4402	0.0440	0. 6614	1.0576

(i.e., lines 9−18). The upper bound on the complexity of this step is $O(2z\varphi\gamma n)$. In the third step (lines 20−23), $O(z\gamma\varphi)$ is required to evaluate solutions constructed in different contexts. Finally, in the last step, the most time-consuming part is selecting the best neighbors of the active user for each target item (i.e., lines 28−35). Specifically, the internal loop (lines 29−33) is repeated up to $\gamma\varphi$ times since the maximum number of nodes in graph $\mathrm{TG\_Best}^c_u(t)$ is equal to $\gamma\varphi$ (each ant constructs a path of length φ). In a specific context c, the upper bound time complexity of the last step is $O(|c|\,\gamma\varphi)$. Thus, $O(m\gamma\varphi)$ is required for selecting the best neighbors of the active user in different contexts.

To sum up, the overall complexity of Algorithm 1 is $O(mn + 2z\varphi\gamma n + m\gamma\varphi)$. However, the actual complexity is smaller because this algorithm iterates over target clusters instead of all clusters. In addition, it should be noted that z is a small constant factor. The reason is that the large number of clusters results in fewer items within each cluster, and when the size of clusters is too small, there is not enough information to infer users' preferences in each context. Thus, the system becomes unable to accurately predict the implicit trust values due to data sparsity (refer to Section 4.1). φ is also a small constant factor since longer propagation levels result in less accurate trust predictions (refer to Section 4.1). Therefore, if $\gamma << n$, then the complexity of Algorithm 1 is of $O(mn + n + m) \approx O(mn)$. In the worst case, when $\gamma = n$, the upper bound complexity of Algorithm 1 is $O(mn + n^2 + mn) \approx O(mn + n^2)$. Hence, the number of ants poses a tradeoff: by increasing γ, the accuracy of predictions is expected to increase, but efficiency (in terms of the execution time) of the algorithm is decreased. The proper value of γ is selected by performing sensitivity analysis (see Section 4.1).

In summary, with $m = O(n)$, similar to the classical user-based CF, STARS requires a time complexity of $O(n^2)$ in order to generate a list of recommendations for an active user at each time step in the online phase. Bearing in mind that STARS considers both dynamic trust and similarity information to adapt itself to dynamically changing user interests, and noting that there is always a trade-off between higher accuracy and better runtime, this time complexity seems reasonable. Our experimental results support this claim. As will be shown in the following sections, STARS significantly improves the quality of recommendations and mitigates the data sparsity, cold-start and MIMC issues, while its runtime is only a little higher than the other counterparts.

4 Experimental results and discussions

In this section, we compare the performance of STARS against some baseline CF algorithms, including UCF [43] and ICF [61]. In recent years, many researchers have exploited these algorithms as benchmarks to evaluate their proposed recommendation approaches. To further evaluate the performance of STARS, we also compare its results with those of TARS [36] and TSF [14], which are the closest approaches to ours. In each baseline algorithm, after predicting the ratings for the test user, we produce a top-N recommendation list composed of those N items having the highest predicted ratings. All approaches are implemented in Matlab and their parameters are tuned to achieve the best results. In particular, to calculate semantic similarities in STARS, we use OWL API, a high level Application Programming Interface (API) for working with OWL ontologies. OWL API has been implemented in Java. It is available as open source under an LGPL license [62]. We perform our experiments on a 2.53 GHz Intel core i5 processor, with 4.00 GB RAM, running Windows 7.

We perform experiments with two real-life data sets: 1) the MovieLens 100K data set which consists of 100,000 ratings from 943 users on 1,682 movies, denoted as ML-100K and 2) the MovieLens 1M data set with about 1 million ratings for 3,952 movies by 6,040 users, denoted as ML-1M. In both data sets, ratings vary from 1 to 5. In our experiments, the empty cells (i.e., unrated items) in the rating matrix are

filled with zero value. The sparsity level of the ML-100K and ML-1M data sets is about 0.937 and 0.958, respectively. To incorporate temporal information in our experiments, we use timestamp of ratings. Timestamps are represented in seconds in both data sets. For ML-100K and ML-1M, ratings have been issued over periods of 215 days and 1039 days, respectively. We round timestamp for each rating to the corresponding day number indexed from 1 to 215 for ML-100K, and 1 to 1039 for ML-1M. In the first data set, starting at time $t_0 = 60$ (i.e., 60 days after the first rating), the system is updated at every $\mu = 23$ days. For ML-1M, the starting time is set to $t_0 = 110$ and the system is updated at every $\mu = 49$ days. Therefore, these data sets allow for 7 and 19 temporal updates, respectively.

At each time t, based on all ratings made prior to t (i.e., training data), we aim to predict the ratings of a test user for target items rated by her before the next update. A top-N ranked list of items is recommended to the test user at time t. Time t is then incremented by μ and the process is repeated. Therefore, at each time step, ratings issued in the previous time step are incorporated into the training set. This set up is consistent with the actual operation of a recommender system in which the training set is incrementally augmented with ratings in the order that users issue them.

To measure the prediction accuracy of the proposed approach in comparison with others, we employ the most well-known related metric, i.e., the standard Mean Absolute Error (MAE) [63]. For each test user u, MAE measures the average absolute deviation between her actual rating $r_{u,i}$ and the predicted rating $\hat{r}_{u,i}$:

$$\text{MAE} = \frac{\sum_{u,i} \left| r_{u,i} - \hat{r}_{u,i} \right|}{|B|} \tag{22}$$

where B denotes the set of ratings to be predicted before the next update. Lower MAE values indicate better prediction accuracy.

In order to evaluate the recommendation quality, we adopt the testing methodology presented in [64]. Let Y be the set of relevant items for test users at time t (i.e., all items that have been rated 5-stars by test users between time t and $t+\mu$). As mentioned before, the training set TR contains all ratings made prior to t. According to the adopted methodology [64], we first train the system over the ratings in TR. Then, for each target item i rated 5-stars by user u in Y, we randomly select 1000 additional items unrated by user u before time $t+\mu$. Then, the ratings for the test item i and for the additional items are predicted and a top-N ranked list of items is recommended. If the target item i is among the N recommended items, then we have a *hit*. The recommendation quality is measured in terms of the recall measure. For any single test case, recall can take either value 0 (in case of miss) or 1 (in case of hit). The overall recall is computed by averaging over all test cases [64]:

$$\text{recall} = \frac{\#hits}{|Y|} \tag{23}$$

where $\#hits$ is the number of test cases in which the algorithm successfully recommends the test item. A drawback of recall, that has been also called Hit-Rate (HR) [44], is that all hits are considered equally regardless of where they occur in the top-N list. This limitation is addressed by the Average Reciprocal Hit-Rank (ARHR) measure [44], which is defined as:

$$ARHR = \frac{1}{|Y|} \sum_{h=1}^{\#hits} \frac{1}{pos_h} \tag{24}$$

where $pos_h \in [1, N]$ is the position of the test item in the ranked recommendation list for the h-th hit. Therefore, in our experiments, we measure the quality of recommendations by looking for the number of hits and their positions within the top-N lists generated by the under comparison algorithms. Higher HR and ARHR values indicate better qualitative performance.

The following parameters have been used at the different stages in our proposed approach: the number of clusters (z, default value 10), the number of ants (γ, default value 100), the relative weight of pheromone trail (α, default value 2), the relative weight of heuristic value (β, default value 1), the maximum trust propagation distance (φ, default value 2), the maximal number of the best trust paths (bpn, default value 50), the pheromone decay value (ρ, default value 0.1), and the time decay weight (θ, default value 0.2). Since STARS relies on a random number generator for the probabilistic selection of nodes during solution construction, each experiment is repeated 10 times and results are averaged over all independent runs to reduce the random variation of the proposed approach. In our experiments, the size of the recommendation list is set to $N = 10$.

4.1 Sensitivity analysis for STARS

In this section, we test the effects of different parameters on the performance of STARS. These parameters include the number of item clusters (z), the number of ants (γ), the relative weight of pheromone trail (α), the relative weight of heuristic value (β), the maximum trust propagation distance (φ), the maximal number of the best trust paths (bpn), the pheromone decay value (ρ), and the time decay weight (θ).

For each particular data set, proper values of the parameters are selected by performing sensitivity analysis on that data set: in order to assign a proper value to each parameter, the parameters are examined individually, i.e., by changing the value for a given parameter while keeping default

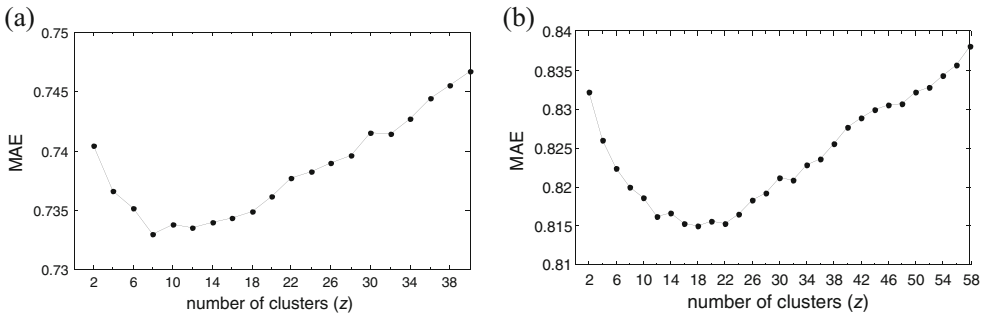


Fig. 3 The effect of parameter z on the prediction performance for **a** ML-100K and **b** ML-1M

values for the rest of parameters. Considering the prediction accuracy measurement, the appropriate value for the given parameter is selected according to the average of the results obtained over all time steps. The examined parameter is given this value and the procedure is repeated for the next parameter until all parameters are assigned with the optimal values.

Effect of parameter z We investigate the impact of different number of clusters on the prediction performance of STARS. Figure 3a and b illustrate MAE under different values of z for the ML-100K and ML-1M data sets, respectively. For the ML-100K data set (Fig. 3a), the average MAE is improved as the value of z increases up to a certain point. Afterwards, the prediction accuracy is degraded as the value of z increases. A similar scenario can also be seen in the ML-1M data set (Fig. 3b). This is because the small number of clusters makes the cluster information too general; therefore, differences among dissimilar items cannot be represented properly. As the number of clusters increases, the number of co-rated items between two users in a cluster decreases. So, for large values of z, there are not enough items in each cluster to derive the implicit trust values between users, and thus, the accuracy of predictions decreases. This observation demonstrates the importance of selecting an appropriate number of item clusters. As shown, for the ML-100K and ML-1M data sets, the best performance is attained for $z = 8$ and $z = 18$, respectively. In the remaining experiments, we keep these numbers as the default values.

Effect of parameter γ To select a proper value for the number of ants in the ML-100K data set, we set the value of γ from 50 to 943 stepped by 50. The maximum number of ants is equal to the number of nodes (i.e., users) in the trust network. For the ML-1M data set, the value of γ varies within a range of 100 to 6040 stepped by 200. Figure 4 illustrates the effect of the number of ants on the prediction performance of STARS. As shown in Fig. 4a, the prediction quality improves as we increase γ. In fact, for small values of γ, there are not enough ants to explore areas in the search space. However, for $\gamma > 250$, the rate of improvement decreases. The reason is that when γ is high, the degree of overlap between solutions constructed by different ants increases. Thus, for high values of γ, the resulting neighborhoods will be similar, and consequently, the predictions are nearly identical. On the other hand, by increasing the number of ants, we harm the efficiency (in terms of execution time) of STARS in its online phase. Thus, to have both effectiveness and efficiency, a prerequisite is the fine-tuning of the number of ants. Regarding efficiency, we measure the run-time of STARS in the online phase by varying the number of ants. Figure 5 shows the average time required to generate the top-10 recommendations for a test user under different values of γ. As the number of ants increases, STARS needs more execution time because more search is done. For the ML-100K data set, the best combination of effectiveness and efficiency can be attained by selecting γ between 250 and 350, where the resulting MAE is about 0.69 (very close to the results when having 943 ants) and run-time is under 0.15 second. Actually, for $\gamma > 350$,

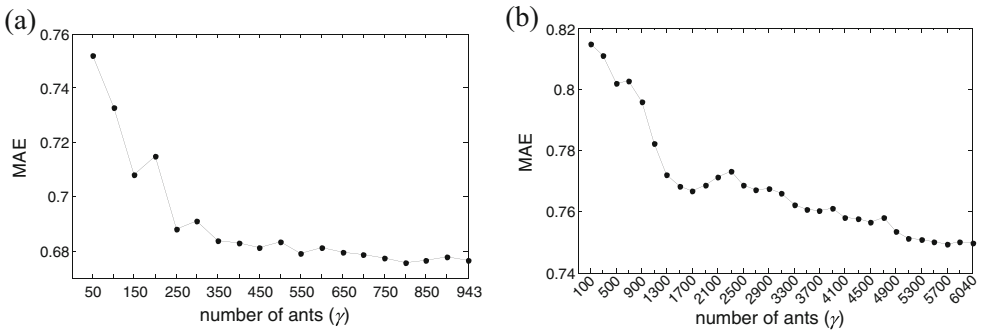


Fig. 4 The effect of parameter γ on the prediction performance for **a** ML-100K and **b** ML-1M

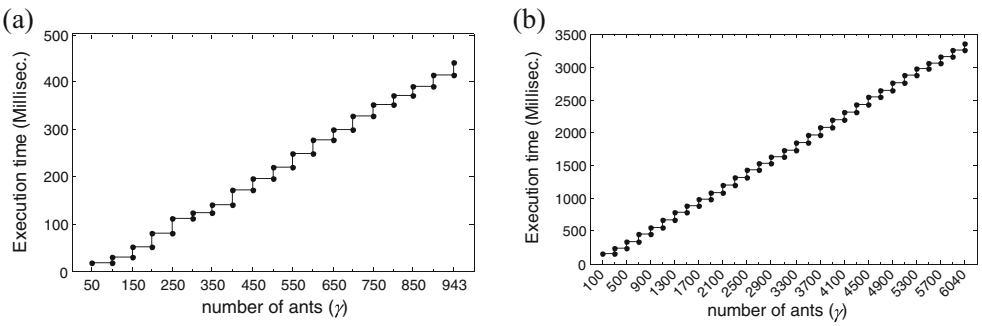


Fig. 5 The effect of parameter γ on the efficiency of STARS for **a** ML-100K and **b** ML-1M

the prediction error decreases slightly while the execution time increases significantly. In the latter experiments, we fix $\gamma = 250$ on ML-100K. Similar results are observed on ML-1M data set, where the best value for γ is about 1300.

Effect of parameters α and β An important step of the STARS approach is to compute the probability of selecting a node based on the trust value and rating similarity (see (12)). Two exponential parameters in (12), namely α and β, have a significant impact on the effectiveness of the STARS approach. When $\alpha \geq 1$ and $\beta = 0$, the probability of selecting a node completely depends on the trust value. When $\alpha = 0$ and $\beta \geq 1$, the trust value is not considered and the transition probability is totally dependent on the rating similarity. To find reasonable values for α and β, we analyze how the combination of these parameters affects the accuracy of predictions in STARS.

We conduct an experiment in which the values of α and β vary within the range of 0 to 10 with step 1. The setting resulting in the best performance will be adopted for the latter experiments of this research. Figure 6 illustrates the effect of parameters α and β on the prediction performance of STARS in both data sets. The results show that considering both trust information and rating similarities can significantly improve the performance compared to the cases that either α or β is equal to zero. When $\beta = 0$, STARS selects neighbors only based on their reputation obtained from previous time steps, and therefore, it cannot adapt itself to dynamically changing user interests. When $\alpha = 0$, STARS only utilizes the observed user-item ratings, leading to poorer performance, especially when dealing with cold users. So, we conclude that both trust information and rating similarities should be combined to improve the recommendation performance.

Figure 6 shows that the best prediction accuracy for the ML-100K data set is achieved by setting $\alpha = 1$ and $\beta = 2$, where the optimal MAE is equal to 0.6852. For the ML-1M data set, these parameters should be set as $\alpha = 1$ and $\beta = 3$, resulting in MAE = 0.7686. According to Fig. 6, in most cases, the lower MAE values are obtained when $\alpha \geq \beta$. More specifically, for ML-100K (ML-1M), MAE is less than 0.687 (0.770) when $\alpha \geq \beta$ in approximately 80 % (67 %) of cases. Based on these results, we may conclude that similarity is more important than the trust information in determining users' neighborhoods in the proposed system. This is due to the reason that the similarity information reflects the current preferences of users, while the trust information reflects the previously learned knowledge based on the global reputation and involvement of users in generating recommendations in the past. By giving more weight to the rating similarity, STARS will be able to better adjust dynamic trust values according to the changes in users' preferences.

Effect of parameter φ In this experiment, we intend to investigate the effects of different maximum trust propagation distances on the prediction performance of STARS. We set the value of φ from 1 (i.e., no propagation) up to 10 since the further away a user is from the source user, the less reliable is the inferred trust value. The effect of different values of φ on the prediction performance is illustrated in Fig. 7. By increasing φ, the accuracy of predictions increases as

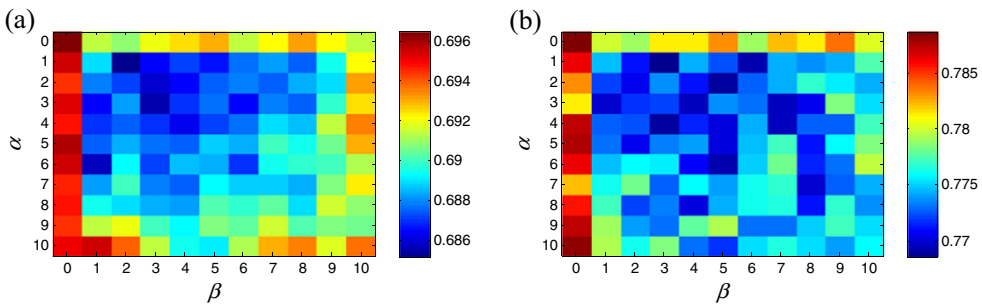


Fig. 6 The effect of parameters α and β on the prediction performance for **a** ML-100K and **b** ML-1M

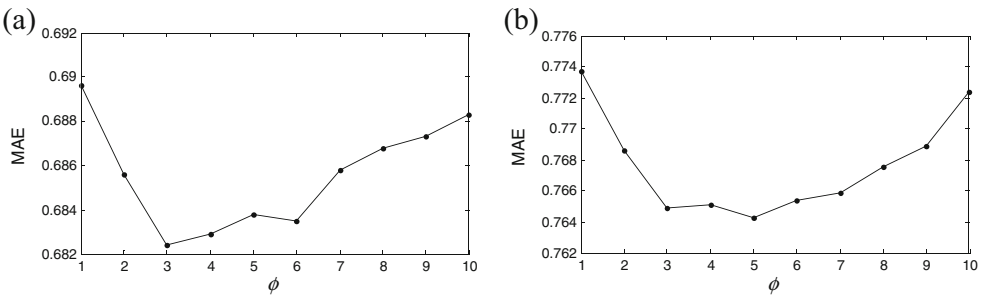


Fig. 7 The effect of different values of φ on the prediction performance for **a** ML-100K and **b** ML-1M

expected. The reason is due to the fact that increasing the propagation distance allows STARS to overcome data sparsity and provide more trusted neighbors for every single user. Nevertheless, for large values of φ, accuracy decreases because when longer propagation levels are used, we are heading further away from the active user, and consequently, the inferred trust value is less reliable. So, involvement of less trustworthy users in predicting the missed ratings has a negative impact on the accuracy of predictions. As shown in Fig. 7, the best performance is obtained when $\varphi = 3$ for ML-100K, and when $\varphi = 5$ for Ml-1M. In the remaining experiments, we keep these values as the default values.

Effect of parameter *bpn* STARS requires to select a set of best trust paths to identify the active user's neighborhood. In this experiment, we intend to determine the best value for parameter *bpn* (i.e., the maximal number of the best trust paths). We set the value of *bpn* from 10 to 590 stepped by 20 for ML-100K, and from 10 to 3170 stepped by 40 for ML-1M. The experimental results for both data sets are displayed in Fig. 8. As shown, small values of *bpn* result in low prediction accuracy since the identified neighbors of the active user are not enough to make accurate predictions. Also, the large values of *bpn* have a negative impact on the accuracy, as solutions with a low path trust value may be considered for neighborhood formation. The best prediction accuracy for the ML-100K data set is achieved by setting $bpn = 230$, where the resulting MAE is equal to 0.6685. For ML-1M, *bpn* should be set to 970, which results in the best prediction performance (i.e., MAE = 0.7469) for this data set. These settings will be adopted for the latter experiments of this paper.

Effect of parameter ρ Here, we investigate the influence that the pheromone decay value has on the performance of STARS. We run experiments with different settings for the evaporation rate. In these experiments, the values of ρ vary from 0 to 0.8 with step 0.01 for both data sets. The effect of parameter ρ on the prediction performance of STARS is illustrated in Fig. 9. The results on both data sets show that when the value of ρ is too small (e.g., $\rho < 0.05$) or too large (e.g., $\rho > 0.5$), the prediction accuracy is low. A very low evaporation rate prevents STARS from being exploratory enough, and thus, results in a fast convergence towards selected neighborhoods in the initial iterations. This makes STARS adapt itself to dynamically changing user interests more slowly since the probability of selecting new neighbors is low and STARS cannot quickly direct its search toward new directions. On the other hand, a very high rate of evaporation decreases the capability of the ants in exploiting the knowledge learned in the previous iterations. As shown in Fig. 9, for ML-100K, the performance of STARS will be more stable when $0.09 \leq \rho \leq 0.16$. For the other data set, the best results are obtained when $0.11 \leq \rho \leq 0.19$. In the following experiments, parameter ρ is set to 0.11 and 0.15 for the ML-100K and ML-1M data sets, respectively.

Effect of parameter θ: Finally, we study the impact of the time decay weight on the prediction quality of STARS. This

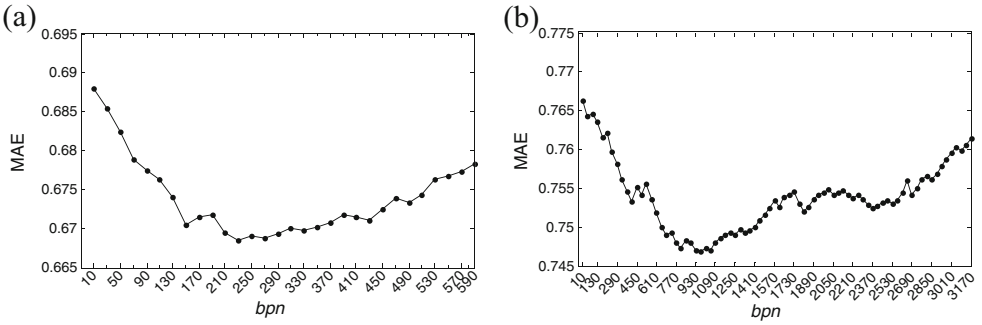


Fig. 8 The effect of different values of *bpn* on the prediction performance for **a** ML-100K and **b** ML-1M

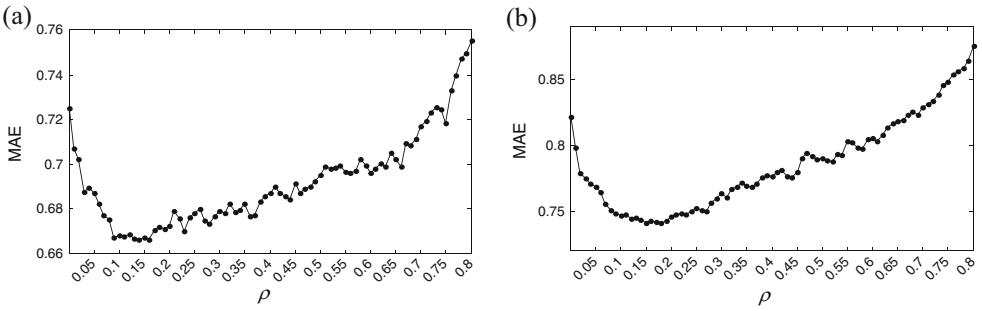


Fig. 9 The effect of different values of ρ on the prediction performance for **a** ML-100K and **b** ML-1M

parameter controls the decay rate for the temporal relevance function when computing similarity (9). In this experiment, we investigate the performance of our approach versus different values of θ. For this purpose, we set the value of θ from 0 to 0.4 with step 0.02. As shown in Fig. 10a, increasing θ to emphasize more on recent ratings improves the prediction quality for the ML-100K data set. However, when $\theta \geq 0.2$, the prediction error increases as θ increases. Actually, for large values of θ, the system almost forgets the historical information compared to more recent data. In other words, placing too much emphasis on the recent ratings and de-emphasizing the history of past behaviors will lead to a lower prediction accuracy. Based on the results presented in Fig. 10a, the optimal value of parameter θ for ML-100K is between 0.1 and 0.18. For the ML-1M data set, similar findings are achieved with optimal values of parameter θ in the interval 0.08 to 0.14. We set $\theta = 0.12$ for ML-100K and $\theta = 0.08$ for the other data set in the remaining experiments.

4.2 The effectiveness of STARS

In this section, we conduct a number of experiments to examine the effectiveness of the proposed approach over the benchmark algorithms, in terms of the prediction, classification and rank accuracy. The experiments also investigate the ability of STARS to resolve the main limitations of CF.

4.2.1 Comparing the accuracy of STARS with baseline algorithms

In this experiment, we compare the accuracy of STARS against all benchmark algorithms at different time slots on both data sets. For this purpose, we select those users who have rated more than 5 items at time $t = t_0$ as the test users and measure the accuracy of different algorithms over a period of time. For ML-100K and ML-1M data sets, the number of users who expressed more than 5 items at time $t = t_0$ is 363 and 2330, respectively (almost 40 % of the users). It should be noted that the system performance for cold users with less than 5 ratings is demonstrated in the next section when we evaluate the effectiveness of STARS in mitigating the cold-start problem. As mentioned before, we use MAE to evaluate the prediction accuracy; also, recall and ARHR are used to evaluate the quality of the top-N recommendations The average MAE, recall and ARHR for each algorithm over different time slots are presented in Fig. 11a–f.

It is not surprising to see that the TSF and TARS algorithms which exploit additional sources of knowledge perform better than the traditional user-based and item-based CF algorithms. However, STARS outperforms all other baseline algorithms for all the cases, indicating that it is effective for improving both the prediction accuracy and quality of recommendations. To have a better view of

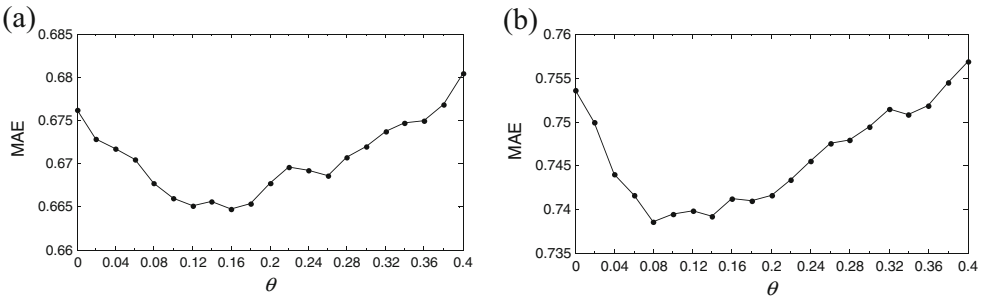
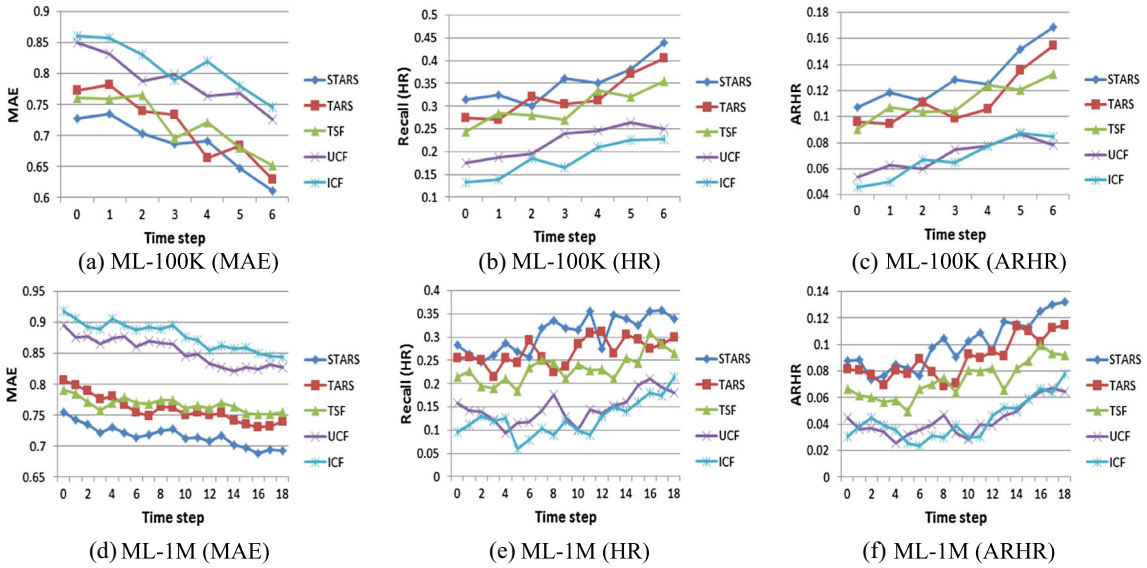


Fig. 10 The effect of different values of θ on the prediction performance for **a** ML-100K and **b** ML-1M



(a) ML-100K (MAE) (b) ML-100K (HR) (c) ML-100K (ARHR)

(d) ML-1M (MAE) (e) ML-1M (HR) (f) ML-1M (ARHR)

Fig. 11 The accuracy of STARS against baseline algorithms for ML-100K and ML-1M

the overall performance of STARS, we further compute the average percentage of improvement obtained by STARS compared to other algorithms across different time steps. Considering an accuracy measure M (such as MAE, recall, etc.), we compute the percentage of improvement in each of the time slots as follows[3]:

$$\text{Improvement} = \frac{\text{STARS} \cdot M - \text{Baseline algorithm} \cdot M}{\text{Baseline algorithm} \cdot M} \times 100\ \% \tag{25}$$

where STARS $\cdot M$ refers to the accuracy of STARS in terms of the M measure, and "Baseline algorithm" refers to any of the algorithms against which we intend to compare the proposed approach. Table 14 shows the results in terms of the MAE, recall and ARHR measures. These results indicate the positive effect of considering all the main properties of trust on the performance of a trust-based recommender system. STARS performs better than its trust-based competitors since it works by traversing context-specific trust networks in a depth-first manner, using both the previously learned trust knowledge and the information about users' current preferences, to select the best neighbors of the active user at each time slot.

[3]The formula can be referred to as the percentage change defined in http://www.math.umb.edu/~joan/MATHQ114/change.htm

4.2.2 Handling the cold start problem

In this section, we investigate the effectiveness of STARS in dealing with cold users. For this purpose, we conduct two classes of experiments. In the first set of experiments, we choose users who rated no items at time $t = t_0$ as the test users and measure the accuracy of generated recommendations for each test user over different time slots. As time goes by, cold users will provide more information to the system, and therefore, users' preferences can be extracted more precisely. For ML-100K and ML-1M data sets, the number of users who expressed no ratings at time $t = t_0$ is 574 and 3698, respectively (almost 61 % of the users).

In the second set of experiments, the users who have rated less than 5 items (excluding users who rated no items) at time $t = t_0$ are chosen as the test users. Since in both data sets, there is not an adequate number of users with less than 5 ratings, we need to simulate such cold-start situation. For this purpose, we partition the users into test users (cold users) and train users using 5-fold cross-validation. Then, we randomly select less than 5 ratings from each test user at time $t = t_0$ as the known ratings for this user at the first time slot. Finally, similar to the first set of experiments, we measure the performance of each algorithm in generating valuable recommendations for each test user over different time slots.

Figure 12 shows the results of our experiments conducted to demonstrate the effectiveness of STARS in dealing with the extreme cold users (i.e., users who rated no items) on both data sets. In the first time slot, since there is no

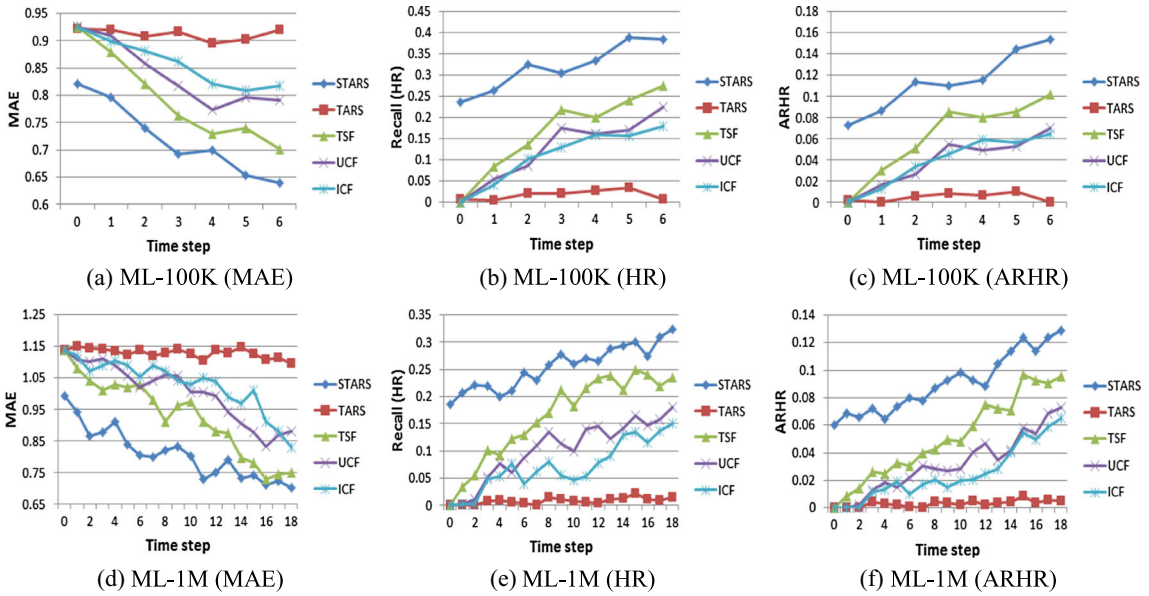
Table 14 The average improvement of STARS over other algorithms in terms of the prediction, classification and rank accuracy

Measure	Data set	Baseline algorithms			
		ICF (%)	UCF (%)	TSF (%)	TARS (%)
MAE	ML-100K	15.48	13.11	4.62	4.08
	ML-1M	18.43	16.02	6.61	5.67
Recall (HR)	ML-100K	92.27	58.57	18.15	9.33
	ML-1M	146.07	108.69	31.95	14.24
ARHR	ML-100K	90.37	84.76	16.40	14.63
	ML-1M	133.23	131.34	36.70	12.10

information about the test users' preferences, UCF, ICF and TSF approaches are unable to predict missing ratings and make any recommendation. As expected, the overall performance of these algorithms tends to improve as the number of items rated by test users increases. As it is clear from Fig. 12, TARS performs poorly for extreme cold users and its performance does not change significantly over time. The reason is that when a user has not provided any ratings, TARS assigns zero weight for all outgoing links of that user in the trust network. Consequently, the probability of selecting any node will be equal to zero, according to the probability equation used in TARS [36]. Therefore, direct neighbors of an extreme cold user are selected randomly since TARS chooses neighbors in the descending order of their associated probabilities. This leads to the poor performance of TARS when we have extreme cold-start settings. Furthermore, in such extreme cases, the weights of edges connecting a cold user and her neighbors remain zero across all time slots, according to the update rule used in TARS [36]. Therefore, this approach cannot utilize the ratings which are gradually provided by the extreme cold users, and the neighbors of such users are always selected randomly in all time slots.

As shown in Fig. 12, STARS consistently achieves significant better performance than its competitors in terms of both prediction accuracy and recommendation quality. In the first time slot where there is no rating information about test users' preferences, STARS can leverage the global trust information for generating valuable recommendations. As the time passes, it tends to generate better recommendations by continuous updating of local trust values between users. We now compare STARS with TSF, which is the second best algorithm for both data sets: considering the ML-100K data set, the average improvements in terms of HR, ARHR and MAE are about 93 %, 82 % and 9 %, respectively; for the ML-1M data set, the average improvements in terms of HR, ARHR and MAE are about 56 %, 78 % and 13 %, respectively.

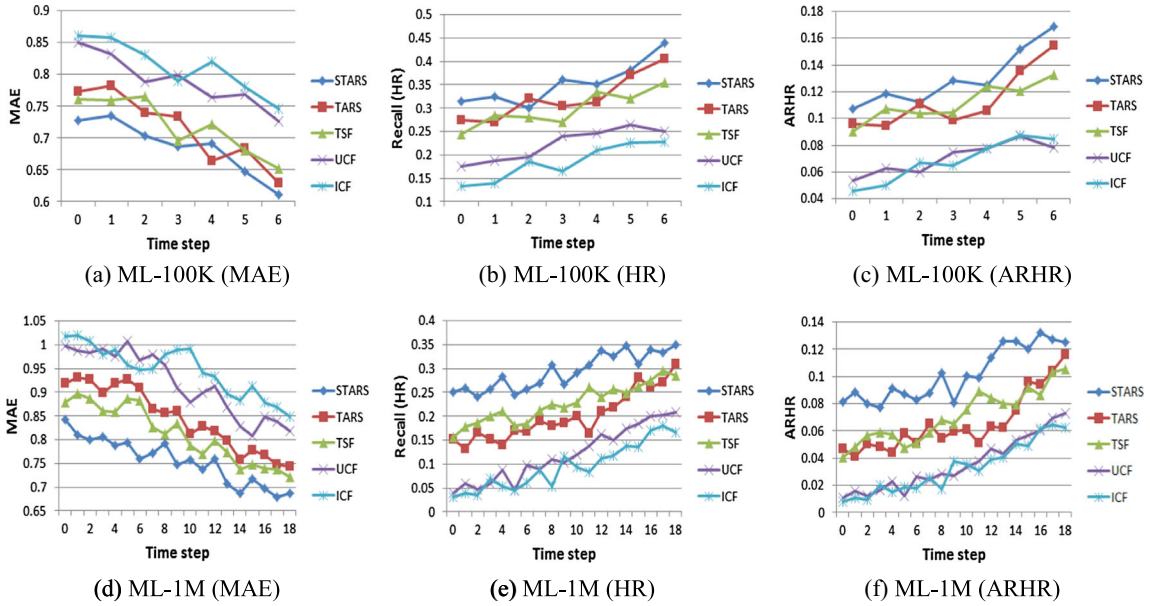
Figure 13 shows the performance of the previously mentioned algorithms in dealing with cold users who rated at least one but less than 5 items at time $t = t_0$. From Fig. 13, we observe that STARS outperforms all other baseline



(a) ML-100K (MAE) (b) ML-100K (HR) (c) ML-100K (ARHR)

(d) ML-1M (MAE) (e) ML-1M (HR) (f) ML-1M (ARHR)

Fig. 12 The performance of different algorithms in handling extreme cold users at time $t = t_0$ on ML-100K and ML-1M



(a) ML-100K (MAE) (b) ML-100K (HR) (c) ML-100K (ARHR)

(d) ML-1M (MAE) (e) ML-1M (HR) (f) ML-1M (ARHR)

Fig. 13 The performance of different algorithms in handling cold users who rated at least one but less than 5 items at time $t = t_0$ on ML-100K and ML-1M

algorithms over all time slots, indicating the effectiveness of STARS in cold conditions. For both data sets, the improvements achieved by STARS over other algorithms tend to be higher in the initial time slots.The reason is that as time goes by, data sparsity decreases, and thus, the difference between the performance of STARS and other benchmarks decreases as well. Table 15 reports the average improvement of STARS over its trust-based competitors, TSF and TARS, according to the results presented in Fig. 13.

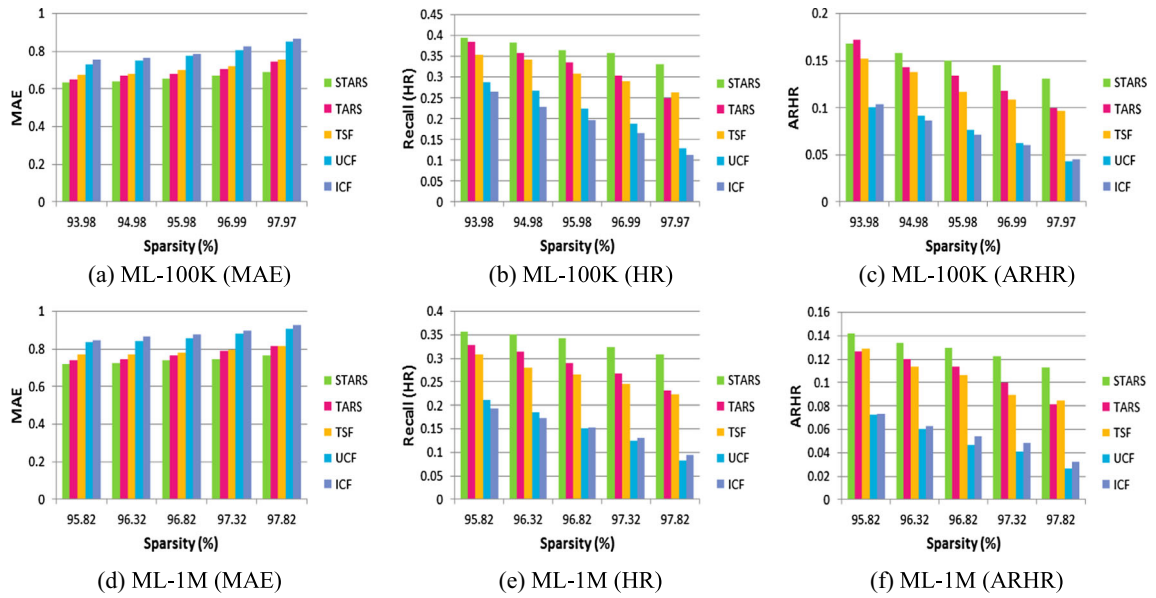
4.2.3 Handling the data sparsity problem

This section examines the effectiveness of STARS in alleviating the data sparsity problem. For this purpose, we design a set of experiments in which the performance of each baseline algorithm is investigated in the last time slot. More specifically, ratings in the last time step are used as the test data in these experiments due to the following reason. When we evaluate the performance of an algorithm over different time slots, the training and test set change over time and the sparsity of training data decreases. To better illustrate the impact of the data sparsity problem, we fix the test set and represent the performance of each algorithm only in the last time slot. With a fixed test set, we evaluate the performance of each algorithm on different sparsity levels by randomly choosing different subsets of the rating data from the training set.

For the ML-100K and ML-1M data sets, the test set consists of 4582 and 2272 ratings, respectively. The density of the training data *TR* (i.e., ratings issued in the remaining time steps) is 6.02 % and 4.18 %, respectively. The sparsity level is defined as (1 − density), where density = #Ratings / (# Users × # Items). Thus, the sparsity level of the training data for the ML-100K and ML-1M is 93.98 % and 95.82 %, respectively. In order to evaluate the performance of each algorithm at higher sparsity levels, we randomly choose different subsets of the rating data from

Table 15 The average improvement of STARS over its trust-based competitors in cold conditions where only few ratings are available

Measure	Data set	Baseline algorithms	
		TSF (%)	TARS (%)
MAE	ML-100K	8.10	8.12
	ML-1M	7.06	10.69
Recall (HR)	ML-100K	20.48	19.30
	ML-1M	29.71	47.08
ARHR	ML-100K	18.53	18.66
	ML-1M	43.28	55.43



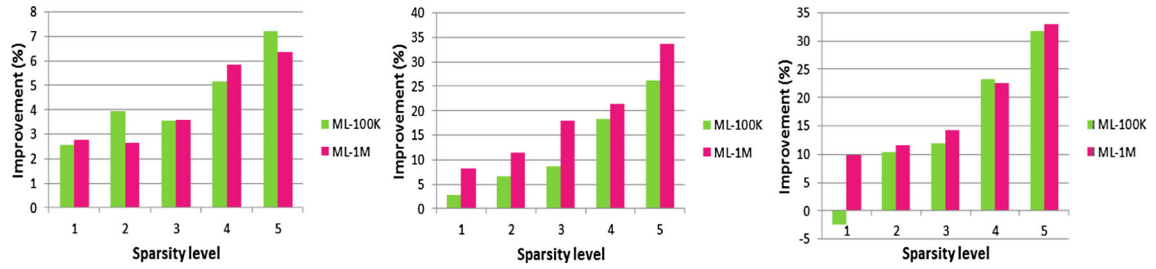
(a) ML-100K (MAE) (b) ML-100K (HR) (c) ML-100K (ARHR)

(d) ML-1M (MAE) (e) ML-1M (HR) (f) ML-1M (ARHR)

Fig. 14 Comparison of STARS with other benchmarks against different sparsity levels on ML-100K and ML-1M

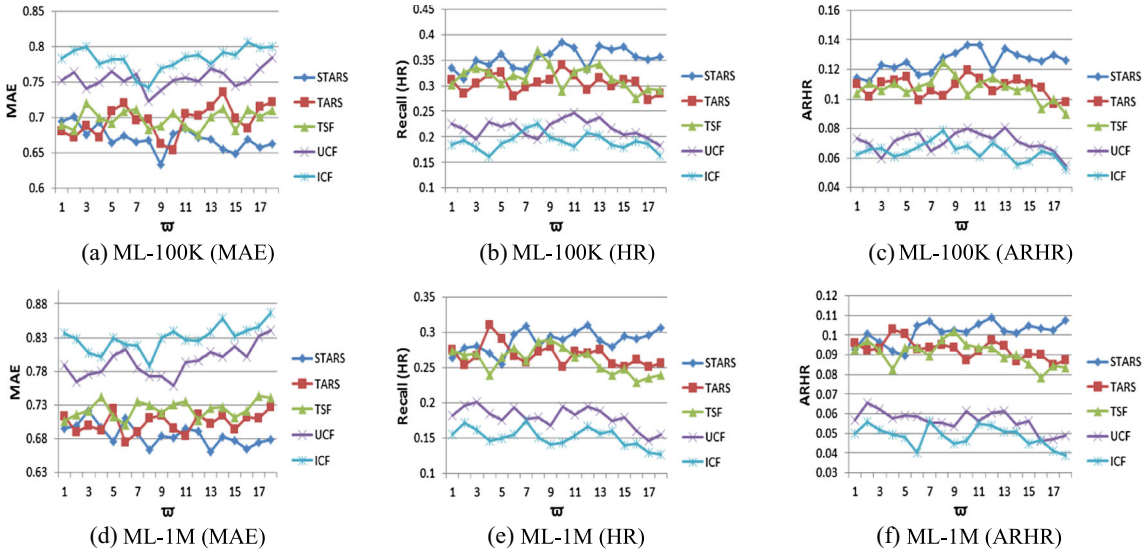
TR for training. Assume that π is the maximum number of observed ratings for each user in the training set. In order to decrease the density of the training data, for each user who has rated more than π items, we randomly pick π observed ratings from *TR* for training. For example, in ML-100K, if we set $\pi = 182$, then the density of the training data decreases from 6.02 % to 5.02 %. To achieve lower levels of density, we set π =113, 69 and 39, leading to different density levels of 4.02 %, 3.01 %, 2.03 % correspondingly. Therefore, with a fixed test set, we vary the sparsity of the training set from 93.98 % to 97.97 % with a step increment of about 1 %. Similarly, For ML-1M, we set $\pi = 435$, 275, 187 and 127, leading to different density levels of 3.68 %, 3.18 %, 2.68 %, 2.18 %, respectively. Therefore, the sparsity of the training set is varied from 95.82 % to 97.82 % with a step increment of 0.5 %.

Figure 14 shows the performance of each algorithm on different sparsity levels for both data sets. As expected, by incorporating an additional source of information, trust-based algorithms outperform traditional user-based and item-based CF algorithms. The transitive property of trust helps in alleviating the data sparsity problem. Among the trust-based algorithms, STARS performs significantly better than TSF and TARS for all sparsity levels. The reason is that the application of the global trust model at the initial stage produces a robust system capable of finding more trusted neighbors for each user even when there is a very sparse data set. As time passes, more accurate neighbors are selected by the local update of dynamic trust pheromone values according to the users' current preferences. To better illustrate the improvement achieved by STARS over the other baseline algorithms as the sparsity increases, Fig. 15 shows the percentage of improvement that STARS results comparing with the second best performing algorithm for both data sets across the five sparsity levels. By Fig. 15, the relative improvement increases as the training set becomes sparser. On the sparsest level, STARS outperforms the second best algorithm by at least 26.22 %, 31.79 % and 6.38 % in terms of HR, ARHR and MAE, respectively.



(a) MAE (b) HR (c) ARHR

Fig. 15 The improvement of STARS over the second best algorithm across the five sparsity levels



(a) ML-100K (MAE) (b) ML-100K (HR) (c) ML-100K (ARHR)

(d) ML-1M (MAE) (e) ML-1M (HR) (f) ML-1M (ARHR)

Fig. 16 The performance of different algorithms in handling the MIMC problem for ML-100K and ML-1M

4.2.4 Handling the MIMC problem

We now turn to investigate the impact of the proposed approach on the MIMC problem. For simplicity, the content of a movie is only described via its associated genres in the experiments of this section. Therefore, we only consider the *belongsToGenre* object property (refer to Section 3.1.1) to compute semantic similarities and cluster items. Both ML-100K and ML-1M data sets provide a same genre set *GS* which consists of 18 different types of genres, including Action, Children's, Horror, Musical, etc. Now, we need to simulate an environment where items have different content. To this end, we gradually increase the number of available genres in the data sets.

Suppose that ϖ is the number of available genres in the data sets. We vary the value of ϖ in the range from 1 to 18 with step 1. At each step, we select ϖ number of genres from *GS* and only keep the movies associated with the selected genres in the data set. For each value of ϖ, we repeat the experiment for all the subsets of size ϖ from *GS*, and then, the results are averaged over all subsets. It is worth noting that the number of clusters (i.e., parameter z) is fine-tuned for each experiment separately.

Figure 16 shows the average performance (over all time slots) of each algorithm against different values of ϖ for both data sets. As shown, the overall prediction accuracy and recommendation quality of STARS tend to improve as we increase the value of ϖ. For both data sets, STARS shows similar or a little worse performance in comparison with the TARS and TSF algorithms when we have small values of ϖ, while by increasing the number of available genres, it outperforms these competitors. The reason is that when the value of ϖ is small (e.g. $\varpi < 5$), there may not be enough data in each cluster to compute the implicit trust values between users. However, by increasing ϖ, the advantage of STARS in utilizing context-dependent trust relations will be more obvious and its performance will tend to increase. By utilizing context-dependent trust relations, trusted neighbors of an active user for a horror movie, for example, may be different from her trustworthy neighbors for a comedy movie. As mentioned in Section 3.1.1, STARS calculates implicit trust values between users based on the semantic features of items. This allows STARS to infer multiple trust relationships between users, each of which may be in a different context. Therefore, the proposed approach is capable of generating proper recommendations in the environments where users have different interests and items

Table 16 The average improvement of STARS over the second best algorithm in dealing with the MIMC problem when $\varpi > 10$

Measure	Data set	Improvement (%)
MAE	ML-100K	4.24
	ML-1M	3.98
Recall (HR)	ML-100K	15.01
	ML-1M	13.18
ARHR	ML-100K	18.98
	ML-1M	14.89

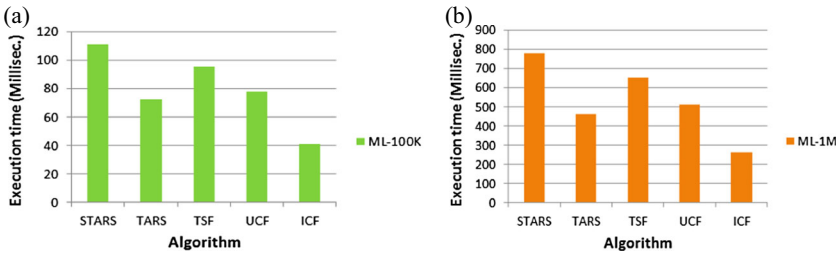


Fig. 17 Comparison of the run-time performance of STARS with other benchmarks for **a** ML-100K and **b** ML-1M

have different content. Based on the results presented in Fig. 16, Table 16 reports the average improvement of STARS over the second best performing algorithm for both data sets when $\varpi > 10$.

4.3 Run-time performance of STARS

In this section, we compare the amount of execution time required for the online phase of different algorithms. Figure 17 shows the average online execution time needed to generate the top-10 recommendations for a test user considering both data sets. All the times in Fig. 17 are in milliseconds. The results show that STARS is the slowest algorithm in comparison with other baseline algorithms. ICF is faster than any other algorithm since it completes the whole task of generating the item-neighborhood matrix in the offline mode. TARS presents the second best performance here. The reason is that it performs a modified breadth-first search in the trust network to generate recommendations. So, it does not need to search the whole database to identify the best neighbors of an active user. TSF needs more time than UCF to provide recommendations because it combines both the user-based trust-enhanced CF and the item-based semantic-enhanced CF into a single framework, which makes it computationally more expensive. From Fig. 17, we can observe that the execution time of STARS is a little larger than the TSF algorithm. This is due to the fact that STARS traverses context-specific trust networks in a depth-first manner, using both the previously learned trust knowledge and time-based user-to-user similarities, to select the best neighbors of the active user. Intuitively, it seems reasonable that such an algorithm will require more time to generate recommendations because there is always a trade-off between higher accuracy and better execution time. Although STARS takes more running time than other baseline algorithms, it achieves significantly effective results in terms of accuracy, especially when dealing with the cold-start, data sparsity and MIMC problems.

5 Conclusions and future work

In this paper, we have made progress towards a more effective trust-based recommender system which takes all distinct properties of trust into account. We have proposed a new dynamic recommender system, STARS, which considers contextual and temporal information for selecting the most trustworthy neighbors of the active user according to her current interests in a specific type of items. Hence, it is able to adapt itself to dynamically changing user interests. STARS has been designed in such a way that it can satisfy asymmetry, transitivity, dynamicity and context-dependency properties of trust. This is achieved by applying the ant colony optimization algorithm on the semantically-enhanced trust relations. In the proposed system, the weights of trust relationships are initialized with the global trust values in a specific context, and then, as time goes by, they are updated locally to reflect the dynamic trust values assigned by one user to another in that context.

Incorporation of both global and local trusts into CF along with the trust computation based on the semantic features of items allows STARS to alleviate the data sparsity, cold-start and MIMC problems. Time-based experiments on two real-world data sets showed that the proposed approach:

(1) has a better overall performance compared to other benchmarks in terms of the prediction, classification and rank accuracy;
(2) achieves significant improvements against other algorithms when dealing with cold users, especially in the case where the active users have not provided any ratings;
(3) can effectively handle the MIMC problem when there are enough items with different content in the data set; and
(4) has a little higher execution time than the other competitors.

In the present work, we have adopted a relatively simple method to compute semantic similarities between items (see (1)). For future research, we intend to consider a more sophisticated semantic inference approach in order to improve the accuracy of semantic similarities used by STARS. Furthermore, one potential limitation of the present work is that we use k-medoids which is a hard clustering method assigning each item to exactly one cluster. However, since an item may have different characteristics, it

can belong to more than one cluster. Therefore, further work direction is towards applying a soft (fuzzy) clustering method which allows items to be members of two or more discovered clusters. In this case, our proposed system must be redesigned such that the process of neighborhood formation is performed by considering multiple clusters instead of exactly one cluster. Another direction of future work is to further improve the accuracy of the proposed approach by incorporating different types of contextual information that may affect trust inference such as the user's current mood, location or activity. Finally, in order to improve the neighborhood formation in STARS, future studies will focus on incorporating more advanced modeling techniques for the analysis of the temporal dynamics of user feedbacks.

Acknowledgments We would like to thank Dr. Belmond Yoberd and Professor Nader Bagherzadeh, University of California, Irvine, for their helpful suggestions in editing the paper.

References

1. Park DH, Kim HK, Choi IY, Kim JK (2012) A literature review and classification of recommender systems research. Expert Syst Appl 39:10059–10072. doi:10.1016/j.eswa.2012.02.038
2. Vozalis MG, Margaritis KG (2007) Using SVD and demographic data for the enhancement of generalized collaborative filtering. Inf Sci 177:3017–3037. doi:10.1016/j.ins.2007.02.036
3. Anand D, Bharadwaj KK (2011) Utilizing various sparsity measures for enhancing accuracy of collaborative recommender systems based on local and global similarities. Expert Syst Appl 38:5101–5109. doi:10.1016/j.eswa.2010.09.141
4. Chen Y, Wu C, Xie M, Guo X (2011) Solving the sparsity problem in recommender systems using association retrieval. J Comput 6:1896–1902
5. Zhang J, Lin Y, Lin M, Liu J (2016) An effective collaborative filtering algorithm based on user preference clustering. Appl Intell:1–11
6. Bobadilla J, Ortega F, Hernando A, Bernal J (2012) A collaborative filtering approach to mitigate the new user cold start problem. Knowl-Based Syst 26:225–238
7. Lika B, Kolomvatsos K, Hadjiefthymiades S (2014) Facing the cold start problem in recommender systems. Expert Syst Appl 41:2065–2073. doi:10.1016/j.eswa.2013.09.005
8. Vizine Pereira AL, Hruschka ER (2015) Simultaneous co-clustering and learning to address the cold start problem in recommender systems. Knowl-Based Syst 82:11–19. doi:10.1016/j.knosys.2015.02.016
9. Zhang Z, Liu H (2015) Social recommendation model combining trust propagation and sequential behaviors. Appl Intell 43:695–706
10. Martín-Vicente MI, Gil-Solla A, Ramos-Cabrer M et al (2014) A semantic approach to improve neighborhood formation in collaborative recommender systems. Expert Syst Appl 41:7776–7788
11. Li Y, Lu L, Xuefeng L (2005) A hybrid collaborative filtering method for multiple-interests and multiple-content recommendation in E-Commerce. Expert Syst Appl 28:67–77
12. Choi K, Suh Y (2013) A new similarity function for selecting neighbors for each target item in collaborative filtering. Knowl-Based Syst 37:146–153
13. Al-Hassan M, Lu H, Lu J (2015) A semantic enhanced hybrid recommendation approach: a case study of e-Government tourism service recommendation system. Decis Support Syst 72:97–109. doi:10.1016/j.dss.2015.02.001
14. Shambour Q, Lu J (2012) A trust-semantic fusion-based recommendation approach for e-business applications. Decis Support Syst 54:768–780
15. Forsati R, Mahdavi M, Shamsfard M, Sarwat M (2014) Matrix factorization with explicit trust and distrust side information for improved social recommendation. ACM Trans Inf Syst (TOIS) 32:17
16. Mobasher B, Jin X, Zhou Y (2004) Semantically enhanced collaborative filtering on the web. In: Web mining: from web to semantic web. Springer, pp 57–76
17. Chen R-C, Huang Y-H, Bau C-T, Chen S-M (2012) A recommendation system based on domain ontology and SWRL for anti-diabetic drugs selection. Expert Syst Appl 39:3995–4006. doi:10.1016/j.eswa.2011.09.061
18. Ruotsalo T, Haav K, Stoyanov A et al (2013) SMARTMUSEUM: a mobile recommender system for the Web of Data. Web Semant Sci Serv Agents World Wide Web 20:50–67. doi:10.1016/j.websem.2013.03.001
19. Capelle M, Hogenboom F, Hogenboom A, Frasincar F (2013) Semantic news recommendation using wordnet and bing similarities. In: Proceedings of the 28th annual ACM symposium on applied computing. ACM, New York, pp 296–302
20. Gohari FS, Tarokh MJ (2015) New recommender framework: combining semantic similarity fusion and bicluster collaborative filtering. Comput Intell. doi:10.1111/coin.12066
21. Bogdanov D, Haro M, Fuhrmann F et al (2013) Semantic audio content-based music recommendation and visualization based on user preference examples. Inf Process Manag 49:13–33. doi:10.1016/j.ipm.2012.06.004
22. Ostuni VC, Di Noia T, Di Sciascio E et al (2015) A semantic hybrid approach for sound recommendation. In: Proceedings of the 24th international conference on world wide web companion. International world wide web conferences steering committee. Republic and Canton of Geneva, Switzerland, pp 85–86
23. Hwang C-S, Chen Y-P (2007) Using trust in collaborative filtering recommendation. In: New trends in applied artificial intelligence. Springer, pp 1052–1060
24. Yuan W, Guan D, Lee Y-K et al (2010) Improved trust-aware recommender system using small-worldness of trust networks. Knowl-Based Syst 23:232–238
25. Lathia N, Hailes S, Capra L (2008) Trust-based collaborative filtering. In: Trust management II. Springer, pp 119–134
26. Papagelis M, Plexousakis D, Kutsuras T (2005) Alleviating the sparsity problem of collaborative filtering using trust inferences. In: Trust management. Springer, pp 224–239
27. Yuan W, Shu L, Chao H-C et al (2010) ITARS: trust-aware recommender system using implicit trust networks. Communications, IET 4:1709–1721
28. O'Donovan J, Smyth B (2005) Trust in recommender systems. In: Proceedings of the 10th international conference on intelligent user interfaces. ACM, pp 167–174
29. Pitsilis G, Marshall LF (2004) A model of trust derivation from evidence for use in recommendation systems. University of Newcastle upon Tyne, Computing Science
30. Guo G, Zhang J, Thalmann D (2014) Merging trust in collaborative filtering to alleviate data sparsity and cold start. Knowl-Based Syst 57:57–68. doi:10.1016/j.knosys.2013.12.007
31. Guo G, Zhang J, Thalmann D et al (2014) From ratings to trust: an empirical study of implicit trust in recommender systems. In: Proceedings of the 29th annual ACM symposium on applied computing. ACM, pp 248–253

32. Massa P, Avesani P (2004) Trust-aware collaborative filtering for recommender systems. In: On the move to meaningful internet systems 2004: CoopIS, DOA, and ODBASE. Springer, pp 492–508
33. Haydar C, Roussanaly A, Boyer A (2013) Local trust versus global trust networks in subjective logic. In: 2013 IEEE/WIC/ACM international joint conferences on web intelligence (WI) and intelligent agent technologies (IAT). IEEE, pp 29–36
34. Dorigo M, Birattari M (2010) Ant colony optimization. In: Encyclopedia of machine learning. Springer, pp 36–39
35. Dorigo M, Maniezzo V, Colorni A (1996) Ant system: optimization by a colony of cooperating agents. IEEE Trans Syst Man Cybern B Cybern 26:29–41
36. Bedi P, Sharma R (2012) Trust based recommender system using ant colony for trust computation. Expert Syst Appl 39:1183–1190
37. Dorigo M, Birattari M, Stutzle T (2006) Ant colony optimization. IEEE Comput Intell Mag 1:28–39. doi:10.1109/MCI.2006.329691
38. M'Hallah R, Alhajraf A (2008) Ant colony optimization for the single machine total earliness tardiness scheduling problem. In: Nguyen NT, Borzemski L, Grzech A, Ali M (eds) New frontiers in applied artificial intelligence. Springer, Berlin, pp 397–407
39. Soleimani-Pouri M, Rezvanian A, Meybodi MR (2012) Finding a maximum clique using ant colony optimization and particle swarm optimization in social networks. In: Proceedings of the 2012 international conference on advances in social networks analysis and mining (ASONAM 2012). IEEE Computer Society, pp 58–61
40. Zecchin AC, Simpson AR, Maier HR et al (2006) Application of two ant colony optimisation algorithms to water distribution system optimisation. Math Comput Model 44:451–468. doi:10.1016/j.mcm.2006.01.005
41. Alba E (2005) Parallel metaheuristics: a new class of algorithms. Wiley
42. Burke R (2002) Hybrid recommender systems: survey and experiments. User Model User-Adap Inter 12:331–370
43. Resnick P, Iacovou N, Suchak M et al (1994) GroupLens: an open architecture for collaborative filtering of netnews. In: Proceedings of the 1994 ACM conference on Computer supported cooperative work. ACM, pp 175–186
44. Deshpande M, Karypis G (2004) Item-based top-n recommendation algorithms. ACM Trans Inf Syst (TOIS) 22:143–177
45. Ma H, King I, Lyu MR (2007) Effective missing data prediction for collaborative filtering. In: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval. ACM, pp 39–46
46. Blanco-Fernández Y, Pazos-Arias JJ, Gil-Solla A et al (2008) A flexible semantic inference methodology to reason about user preferences in knowledge-based recommender systems. Knowl-Based Syst 21:305–320
47. Martín-Vicente MI, Gil-Solla A, Ramos-Cabrer M et al (2012) Semantic inference of user's reputation and expertise to improve collaborative recommendations. Expert Syst Appl 39:8248–8258
48. Lu J, Shambour Q, Xu Y et al (2013) A web-based personalized business partner recommendation system using fuzzy semantic techniques. Comput Intell 29:37–69
49. Ma H (2013) An experimental study on implicit social recommendation. In: Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval. ACM, pp 73–82
50. Yuan W, Guan D, Lee Y-K, Lee S (2011) The small-world trust network. Appl Intell 35:399–410
51. Fang H, Guo G, Zhang J (2015) Multi-faceted trust and distrust prediction for recommender systems. Decis Support Syst 71:37–47
52. Bellaachia A, Alathel D (2012) Trust-based ant recommender (T-BAR). In: 2012 6th IEEE international conference intelligent systems (IS). IEEE, pp 130–135
53. Bellaachia A, Alathel D (2014) DT-BAR : a dynamic ANT recommender to balance the overall prediction accuracy for all users. Academy & Industry Research Collaboration Center (AIRCC):141–151
54. Liu NN, Zhao M, Xiang E, Yang Q (2010) Online evolutionary collaborative filtering. In: Proceedings of the fourth ACM conference on recommender systems. ACM, pp 95–102
55. Kaufman L, Rousseeuw PJ (2009) Finding groups in data: an introduction to cluster analysis. Wiley
56. Carrer-Neto W, Hernández-Alcaraz ML, Valencia-García R, García-sánchez F (2012) Social knowledge-based recommender system. Application to the movies domain. Expert Syst Appl 39:10990–11000. doi:10.1016/j.eswa.2012.03.025
57. MacQueen J et al (1967) Some methods for classification and analysis of multivariate observations. In: Proceedings of the fifth berkeley symposium on mathematical statistics and probability, Oakland, pp 281–297
58. Jin R, Si L (2004) A study of methods for normalizing user ratings in collaborative filtering. In: Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval. ACM, pp 568–569
59. Adomavicius G, Tuzhilin A (2005) Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. IEEE Trans Knowl Data Eng 17:734–749
60. Cormen TH (2009) Introduction to algorithms. MIT press
61. Sarwar B, Karypis G, Konstan J, Riedl J (2001) Item-based collaborative filtering recommendation algorithms. In: Proceedings of the 10th international conference on World Wide Web. ACM, pp 285–295
62. Horridge M, Bechhofer S (2011) The OWL API: a Java API for OWL ontologies. Semantic Web 2:11–21
63. Herlocker JL, Konstan JA, Terveen LG, Riedl JT (2004) Evaluating collaborative filtering recommender systems. ACM Trans Inf Syst (TOIS) 22:5–53
64. Cremonesi P, Koren Y, Turrin R (2010) Performance of recommender algorithms on top-n recommendation tasks. In: Proceedings of the fourth ACM conference on Recommender systems. ACM, pp 39–46

Faezeh Sadat Gohari received her B.S. degree in Software Engineering from University of Science and Culture, Iran, in 2011. She received her M.S. from K.N. Toosi University of Technology, Iran, in 2013. She is currently a Ph.D. candidate in Software Engineering at Shahid Beheshti University, Iran. Her research interests include recommender systems, semantic web, soft computing and data mining.

Hassan Haghighi is an Assistant Professor at the Faculty of Computer Science and Engineering, Shahid Beheshti University, Iran. He received his Ph.D. degree in Software Engineering from Sharif University of Technology, Iran, in 2009. His main research interests include software testing and formal methods, and he has more than 80 journal and conference papers in these areas.

Fereidoon Shams Aliee has received his PhD in Software Engineering from the Department of Computer Science, Manchester University, UK, in 1996 and his M.S. from Sharif University of Technology, Tehran, Iran, in 1990. His major interests are Software Architecture, Enterprise Architecture, Service Oriented Architecture, Agile Methodologies, Ultra-Large-Scale (ULS) Systems and Ontological Engineering. He is currently an Associate Professor at the Faculty of Computer Science and Engineering, Shahid Beheshti University, Iran. Also, he is heading two research groups, namely ASER (Automated Software Engineering Research) (isa.sbu.ac.ir) and ISA (Information Systems Architecture) (isa.sbu.ac.ir) at Shahid Beheshti University.