Expert Systems With Applications 237 (2024) 121541

Contents lists available at ScienceDirect

Expert Systems With Applications

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

A reinforcement learning recommender system using bi-clustering and Markov Decision Process

Arta Iftikhar [a], Mustansar Ali Ghazanfar [b,*], Mubbashir Ayub [a], Saad Ali Alahmari [c], Nadeem Qazi [b], Julie Wall [b]

[a] Department of Software Engineering, University of Engineering and Technology, Taxila, Pakistan
[b] Department of Computer Science & Digital Technologies, University of East London, United Kingdom
[c] Department of Computer Science, AL-Imam Mohammad Ibn Saud Islamic University, Saudi Arabia

ARTICLE INFO

Keywords:
Reinforcement learning
Markov Decision Process
Bi-clustering
Q-learning
Policy

ABSTRACT

Collaborative filtering (CF) recommender systems are static in nature and does not adapt well with changing user preferences. User preferences may change after interaction with a system or after buying a product. Conventional CF clustering algorithms only identifies the distribution of patterns and hidden correlations globally. However, the impossibility of discovering local patterns by these algorithms, headed to the popularization of bi-clustering algorithms. Bi-clustering algorithms can analyze all dataset dimensions simultaneously and consequently, discover local patterns that deliver a better understanding of the underlying hidden correlations. In this paper, we modelled the recommendation problem as a sequential decision-making problem using Markov Decision Processes (MDP). To perform state representation for MDP, we first converted user-item votings matrix to a binary matrix. Then we performed bi-clustering on this binary matrix to determine a subset of similar rows and columns. A bi-cluster merging algorithm is designed to merge similar and overlapping bi-clusters. These bi-clusters are then mapped to a squared grid (SG). RL is applied on this SG to determine best policy to give recommendation to users. Start state is determined using Improved Triangle Similarity (ITR similarity measure. Reward function is computed as grid state overlapping in terms of users and items in current and prospective next state. A thorough comparative analysis was conducted, encompassing a diverse array of methodologies, including RL-based, pure Collaborative Filtering (CF), and clustering methods. The results demonstrate that our proposed method outperforms its competitors in terms of precision, recall, and optimal policy learning.

1. Introduction

The indispensable integration of Recommender Systems (RSs) into our daily lives is undeniable, given their wide spread involvement in diverse activities ranging from social network interactions to shopping decisions and entertainment choices on streaming platforms. While the recommendation problem was historically construed as a classification or prediction challenge, primarily tackled through Collaborative Filtering (CF), the landscape has progressively evolved. Contemporary perspectives frame the recommendation dilemma as a sequential decision-making endeavor, invoking the utilization of Markov Decision Processes (MDP) to formulate recommendations, thus inviting the application of Reinforcement Learning (RL) algorithms. Unlike conventional methods such as CF and content-based filtering, RL exhibits the capacity to address dynamic, sequential user-system interactions, opening new avenues in the realm of recommendation systems. Although the idea of using RL for recommendations is not new, but, due to the scalability problems of traditional RL algorithms, applying RL was not very practical (Afsar, 2022).

Currently, the recommendation problem is primarily solved using techniques that include CF, content-based filtering, and hybrid methods. Although these techniques are successful in providing relevant recommendations, however, several problems exists in these techniques, including absence of user engagement in generating recommendations, cold start, sparsity, scalability, and recommendation quality (Ayub et al., 2019; Ayub et al., 2019; Bobadilla, Ortega, Hernando, & Gutiérrez, 2013; Jannach, Zanker, Felfernig, & Friedrich, 2010). Adopting a sequential recommendation perspective enables us to consider various

* Corresponding author.
E-mail addresses: arta.iftikhar@uettaxila.edu.pk (A. Iftikhar), m.ghazanfar@uel.ac.uk (M.A. Ghazanfar), mubbashir.ayub@uettaxila.edu.pk (M. Ayub), saialahmari@imamu.edu.sa (S. Ali Alahmari), n.qazi@uel.ac.uk (N. Qazi), j.wall@uel.ac.uk (J. Wall).

https://doi.org/10.1016/j.eswa.2023.121541
Received 12 June 2023; Received in revised form 16 August 2023; Accepted 8 September 2023
Available online 15 September 2023

facets, including prolonged user engagement and diverse forms of user-item interactions, encompassing actions such as clicks, purchases, and more. The current recommendation techniques fail to model these factors appropriately (Xin, Karatzoglou, Arapakis, & Jose, 2020). So, casting the sequential recommendation problem as a RL problem is a promising direction. Moreover, recommender systems operate within dynamic environments characterized by shifting user preferences, evolving item popularity, and varying item availability. In this context, Reinforcement Learning (RL) emerges as a fitting option due to its adaptability to dynamic settings. RL possesses the capability to learn and adjust to changes, rendering it well-suited for comprehending such volatile environments. Additionally, the application of RL to the recommendation problem is driven by its adeptness in addressing the exploration–exploitation dilemma. Unlike traditional Collaborative Filtering (CF) RSs, which tend to exclusively exploit existing data for recommendations, RL offers a means to effectively navigate the exploration–exploitation dilemma.

As applying RL directly to a user-item voting database is not scalable as it will result in a very large number of state space. So, we performed bi-clustering on user-item voting database to group similar users and items. Bi-clustering also helped us to maintain a finite number of state space for MDP. The quality of these bi-clusters is measured using the Scaled Mean Square Residual (SMSR) fitness function and these bi-clusters are ordered in a list of descending quality values. The bi-clusters in this list are placed in a fixed size $n \times n$ grid, using the cantor-diagonal order traversal method. The action space contains only four actions i.e. up, down, left and right. The start state in the recommendation process (move within the grid) is determined by ITR similarity measure (Iftikhar, Ghazanfar, Ayub, Mehmood, & Maqsood, 2020). The transition function is not completely deterministic and follows $\in$ -greedy policy while environment learning. The $\in$ -greedy policy allows us to select a random action with uniform distribution over a set of available actions. The reward function is measured as the overlapping of users in the current and prospective next state using the Jaccard similarity measure, with more overlapping resulting in a greater reward value. If the agent is not receiving the recommendation of any new items or gets off the grid, then this state is declared as the goal state of the agent. The objective of the RL agent is to maximize the reward, termed as return, which denotes how much the user is satisfied with the recommended items. The proposed method is also able to explain why users are getting recommendations for certain or all items.

The primary contributions of our proposed approach are as follows:

1. Transforming the conventional Collaborative Filtering (CF) recommendation challenge, which traditionally relies solely on historical user preferences without adapting to evolving user tendencies, into a sequential decision-making problem through the application of Markov Decision Process (MDP).
2. Employing two bi-clustering algorithms, namely Bibit and BiMax, to delineate users with akin localized traits.
3. Mapping the generated biclusters to states within the MDP framework on a Squared Grid (SG).
4. Devising a bi-cluster merging algorithm to amalgamate overlapping biclusters, thereby curbing the number of MDP states.
5. Application of Q-Learning and SARSA techniques on the SG to derive optimal recommendations catered to individual users.
6. Evaluation is performed on two different movies dataset, MovieLens ML-100 K and FilmTrust dataset and effectiveness of proposed method in producing optimal policy for generating recommendations is verified.

The rest of the paper is organized as follows. Section 2 provides a literature review; Section 3 outlines the methodology of our proposed method; Section 4 describes the experimental procedure and evaluation parameters; Section 5 gives experimentation results on ML-100 K and FilmTrust datasets; while, Section 6 concludes our work and highlights some future directions.

2. Literature review

Collaborative Filtering (CF) is one of the most widely used techniques in RSs due to its simplicity and accuracy (Ekstrand, Riedl, & Konstan, 2011). Serving as a method of personalization, it relies on the compilation of user-provided ratings/votes associated with particular products and services. (Chowdhury, 2010). These user votes can be obtained through user-provided feedback. This feedback can be collected through implicit means, such as user clicks on specific items, or through explicit actions, where users rate objects using numerical or starred values. Collaborative Filtering (CF) is a technique for forecasting items for a target user. It considers community votes on items that receive high ratings from community members sharing akin tastes with the target user, under the assumption of the target user's affinity for such items. CF first constructs a user-vote database for items, using these votes for predictions. It subsequently identifies resemblances in user voting history and leverages fellow users' votes to generate predictions and recommendations. This technique has been used by Amazon, iTunes, GroupLens system (Konstan et al., 1997) and Ringo (https://www.ringo.com), etc. and is also called *people-to-people correlation.* One such CF method is Improved PCC weighted with RPB (IPWR), as proposed in (Ayub et al., 2019), which deliberates user Rating Preference Behavior (RPB) while calculating the similarities among users. User RPB is modelled as a function of cosine, taking the target user's average voting value and standard deviation as input values. The user RPB is then combined with an enhanced version of standard PCC. The results of several evaluation metrics prove the superior performance of IPWR (Ayub et al., 2019). An enhancement to the Jaccard index was proposed by (Lee (2017)), encompassing the incorporation of both user vote frequencies and the tally of items co-rated by users. This augmentation facilitates the computation of common items rated by two users, regardless of the specific voting scores assigned. However, it's essential to acknowledge that the voting values for these shared items may span a spectrum from normal to extreme. This adaptation has demonstrated its efficacy in mitigating prominent drawbacks of the conventional Jaccard index, primarily its omission of actual voting values in shared items' assessment.

Another heuristic-based similarity measure method, simultaneously embodies Proximity Significance Singularity (PSS) measures along with a modified Jaccard similarity (Liu, Hu, Mian, Tian, & Zhu, 2014). Traditional PIP similarity (Ahn, 2008) faces serious shortcomings, for instance, it is not normalized and cannot be combined with other similarity measures. The enhanced metric by Liu et al. (2014) integrates Jaccard features, encompassing both absolute voting values and the relative prevalence of shared votes, leading to improved accuracy in predictions. Moreover, the similarity is determined not only by considering the local context only but also by the global preferences of user behaviour. This formalization implies building a similarity measure with a non-linear function based on the initial PIP similarity, which is mainly linear. This new similarity measure, named NHSM (Ahn, 2008), is normalized and can be easily combined with other similarity measures. Besides, this novel similarity measure effectively overcomes the shortcomings and drawbacks of traditional systems (Ahn, 2008). ITR similarity measure (Iftikhar et al., 2020) models the voting vectors of two users and their resultant vector as a triangular shape and then computes the triangle similarity between them. The effectiveness and robustness of IPWR and ITR were vetted in (Fkih, 2022).

Clustering is an unsupervised classification method that groups objects based on similarity, forming distinct clusters where objects within each cluster are more alike than those in other clusters. It deals with finding patterns in an unlabeled dataset. Some of the most prominent application areas of clustering are machine learning, pattern recognition, image analysis, outlier detection and bioinformatics (Prakash, Korostenskaja, Lee, Baumgartner, Castillo, & Bagci, 2017). Clustering is quite often performed in daily life, such as arranging documents into folders based on similarity. Clustering is frequently applied in different fields of science like audio forensics (Malik, 2013), computer vision (RaviPrakash et al., 2020), data mining (Costa & Roda, 2011), bioinformatics (Prakash et al., 2017), stock market trend prediction (Liu & Malik, 2014), and smart cities (Ejaz & Anpalagan, 2019a, 2019b; Hammad & Ludlow, 2016).

Sarwar et al. (Sarwar, Karypis, Konstan, & Riedl, 2002) proposed an algorithm that used the clustering approach to reduce the data into a low-dimensional space and argued that it could overcome scalability and sparsity problems. This approach divides the user-item voting matrix into K non-overlapping partitions, using a variant of the K-Means clustering algorithm, called the Bisecting K-Means clustering algorithm. To identify neighbors for an active user, the process involves examining the cluster containing the active user and then generating recommendations by selecting the most-similarity users (i.e., neighbors) from within that cluster. Khalid et al. (Khalid, Ghazanfar, Azam, Aldhafiri, & Zahra, 2017) proposed a one-pass clustering algorithm (SPOP) that maintains good accuracy and scales well with the arrival of new data in the case of a dynamic environment. This overcame the drawbacks of the K-Means clustering algorithm and claimed to increase the accuracy of RS. In SPOP, users are allocated to distinct hyperspheres according to their distances, with the primary metric being PCC using default votes. When a new hypersphere is created, its radius is set as the average radius of all existing hyperspheres. A training model is built incrementally by sequentially processing new data. The radius of hyperspheres is changed dynamically after a defined number of data points. This proposed K-Means clustering-based RS resolves the scalability issues of traditional RSs. The basic K-Means clustering algorithm selects the initial centroid randomly and the performance of clustering results heavily depends on the selection of these centroids. Zahra et al. (Zahra et al., 2015) proposed improved centroid selection in K-Means-based RSs, which can save cost as well as improve the performance of the system. Their proposed centroid selection methods also improved the accuracy of the system and have the potential to exploit underlying data correlation structures. Conventional clustering algorithms discover global correlations which may not be always desirable. We need such clustering algorithms that can discover local correlations and bi-clustering algorithms can help us here.

Reinforcement Learning (RL) is an algorithmic paradigm enabling machines to learn without any a priori knowledge. By engaging in repeated interactions with the environment and responding to rewards and penalties, RL systems determine optimal policies to achieve objectives (Li, Wang, & Gandomi, 2021). This iterative and multi-objective approach is often structured as episodes, with each iteration representing an episode (Li, Wang, Dong, Yeh, & Li, 2021). Many algorithms have been proposed to solve a RL problem and they can be generally divided into tabular and approximate methods. In tabular methods, RL value functions are denoted as tables. Tabular methods are used when we have a small size of action and state space, hence, finding the exact optimal policy can be determined in a small amount of time. The most popular tabular methods consist of Dynamic Programming (DP), Monte Carlo (MC), and Temporal Difference (TD). DP methods assume a perfect model of the environment and use a value function to search for good policies. Two important algorithms from this class are policy iteration (Bohnenberger & Jameson, 2001; Mahmood & Ricci, 2007) and value iteration (Mahmood & Ricci, 2009). Contrary to DP methods, MC methods only need a sample sequence of states, actions, and rewards from the environment, which could be real or simulated. Monte Carlo Tree Search (MCTS) (Vodopivec, Samothrakis, & Ster, 2017) is an important algorithm from this family. Finally, TD methods are a mixture of MC and DP and do not require a model of the environment, instead, they can bootstrap, having the ability to update estimates based on other estimates (Sutton & Barto, 1998a). From this family, an off-policy method, Q-learning (Watkins, 1989) is very famous. While SARSA (Rojanavasu, Srinil, & Pinngern, 2005), an on-policy method, is also very famous. Since the size of the state space can be enormous in RL, function approximation methods try to discover a good approximate solution which can be identified with limited computational resources. In such methods, a useful tactic is to generalize from previous experiences to unseen states. Many techniques have been proposed for function approximation, including artificial neural networks. Value Penalized Q-learning (VPQ) is proposed in (Gao, Xu, Zhou, Li, Wang, Yuan, & Zhao, 2022), that penalizes the unstable Q-values in the regression target using uncertainty-aware weights. This resulted in a conservative Q-function without the need of estimating the behavior policy. This approach is appropriate for RSs having a large number of items.

To the best of our knowledge, Webwatcher (Joachims, Freitag, & Mitchell, 1997) was the first to use RL to enhance the recommendation accuracy of RSs. Webwatcher modelled the web page recommendation problem as an RL problem and used Q-learning to enhance the accuracy of their basic web RS. Their basic web RS was using TF-IDF, to recommend pages similar to the past interest of the user. Taghipour, Kardan and Ghidary (Taghipour, Kardan, & Ghidary, 2007) extended this hint to recommend personalized web pages to the users by grabbing the state dimensionality problem. They used the N-gram model from the web usage mining literature (Mobasher, Cooley, & Srivastava, 2000) and a sliding window to represent states. (Srivihok & Sukonmanee, 2005) proposed an RL-based tourism recommendation system, composed of two main modules. A personalization learner is used to learn the dynamic and static information of users and a personalization ranking module to produce recommendations using Q-learning. Although their work was novel it was not clear how they handled large state and action spaces. In addition, how rewards were generated and how they are assigned was also unclear. (Mahmood, Mujtaba, & Venturini, 2014), proposed an RL-based conversational RS where Q-learning was used to optimize the policy. To minimize state and action space manageably they used a parametric value. In (Hu, Shi, & Liu, 2017), researchers proposed a state compression model to solve the large dimensionality problem of state space. Their idea was to cluster songs based on similar user preferences and then replace songs with song clusters in the learning phase. A web-based RS was presented by Rojanavasu, Srinil and Pinngern (Rojanavasu et al., 2005). This system was composed of two main modules; the global module is responsible for learning global trends like trending products, while the local module is focused on tracking each user's needs. The system uses a weighted combination of both modules to determine what to recommend next. This system suffers from the scalability problem as it was not made clear how they keep track of all users on a global level. Intayoad, Kamyod and Temdee (Intayoad, Kamyod, & Temdee, 2018) used RL for online clustering, to provide a learning path to students based on their specific needs and characteristics. To minimize the large state space they used an N-gram model. Choi et al. (Choi, Ha, Hwang, Kim, Ha, & Yoon, 2018) proposed a greedy-based algorithm using both Q-learning and SARSA. They used the BiMax (Prelić et al., 2006) and Bibit (Rodriguez-Baena, Perez-Pulido, & Aguilar– Ruiz, 2011) binary clustering algorithms to model the state spaces for MDP. The recommendation problem is modelled as a grid world game problem. However, most of the details of their work are hidden as they did not explain how they transformed binary clusters to state spaces in the grid world and in which order transformation was applied as setting the optimized state spaces is crucial to solving MDP. To select the start state, they computed the Jaccard similarity of the target user with all state spaces. The grid world state having maximum similarity is selected as the start space and the action space used involved up, down, left and right actions. Only two evaluation parameters precision and recall were applied to ML-100K dataset. Scalarized Multi-Objective Reinforcement Learning (SMORL) was presented in (Stamenkovic, Karatzoglou, Arapakis, Xin, & Katevas, 2022) to solve multi-objective recommendation tasks. These multi-objectives involve diversity, novelty and accuracy evaluation parameters. These parameters are somewhat contradictory as increasing diversity and novelty decreases accuracy. The SMORL method acts as a regularizer for bringing appropriate properties into the recommendation model, thus accomplishing a balance between accuracy, diversity and novelty of recommendations. In this work they concluded that combined optimization of these three contradicting objectives is crucial for improving metrics that are tightly correlated with user satisfaction. Work in (Ge, Zhao, Yu, Paul, Hu, Hsieh, & Zhang, 2022) applied pareto concept to solve fairness-utility tradeoff of recommender systems using RL.

(Shani, Heckerman, Brafman, & Boutilier, 2005) were the first one to model the recommendation problem as an MDP problem. They suggested that traditional recommendation generation systems are static and do not adapt to evolving trends. They proved that the recommendation generation problem can be modelled as a sequential optimization problem and thus MDPs can be used more appropriately to model the recommendation generation problem. They solved MDP using a DP approach. Since the model parameters of an MDP-based recommender are unknown and deployment on a real system is very costly, they proposed a predictive model that can provide the initial parameters for MDP. They tested their proposed method on an Israeli online bookstore. A music recommender system proposed by Liebman, Saar-Tsechansky and Stone (Liebman, Saar-Tsechansky, & Stone, 2014) is based on RL. To minimize dimensions, each song is represented as a vector of spectral auditory descriptors, including overall loudness, rhythmic properties, the spectral fingerprint of the song and sound changing over time. The reward function is modelled as the song transition pattern and the listener's preference over individual songs. This system, called DJ-MC, was composed of two main units; a learning unit responsible for initialization, playing songs and getting feedback from the user, and a planning unit responsible for recommending the best song to the user. To reduce the song space, K-Means clustering was used. To solve the top-N recommendation problem (Zou, Xia, Ding, Yin, Song, & Liu, 2019), proposed a method, called Div-FMCTS, which worked in two cyclic stages. In the first cycle, optimal top-N recommendations are heuristically searched in the item space using the MCTS algorithm and those found are generalized using neural networks. To overcome the large item space problem of two methods, problem decomposition and structured pruning are used. Gated recurring units are used to encode the user preference information into states. Lu and Yang (Lu & Yang, 2016) presented an RL-based RS that uses fitted-Q for policy optimization. They reported the *Recurrent Deterioration* phenomenon of RSs where an RS suffers from performance degradation when it is trained based on user feedback from previous recommendations.

In our study, bi-clustering was employed to group similar items and users within a subset of a user-item voting matrix. These clusters were subsequently mapped onto an $n \times n$ squared grid, where each grid cell/state corresponded to a specific bi-cluster. The sequencing of bi-cluster mapping onto the grid was deliberate, as it dictated the user/agent movement on these grid cells/states. A similarity metric was computed between all bi-clusters, guiding their arrangement on the grid so that more similar bi-clusters were positioned in closer proximity. When bi-clusters exceeded the grid size, merging was employed to condense their count. Conversely, if the bi-cluster count was lower than the grid size, substantial bi-clusters were decomposed into sub-clusters. The efficacy of our proposed approach was assessed using publicly available datasets, namely ML-100K and FilmTrust datasets. Comprehensive elaboration of our methodology is presented in the subsequent sections.

3. Methodology

Our proposed method consists of several steps, all of which are explained in the coming subsections.

3.1. Generating binarized data

The user-item votings matrix constitutes a fundamental component within recommendation systems, capturing user preferences and their corresponding votes. This matrix encompasses two distinct value categories. The first entails the voting values, signifying users' evaluations of items, often adhering to a predefined scale, conventionally spanning from 1 to 5. The second category delineates instances where users opt not to provide ratings for specific items, denoted as "Ø". In order to establish a binary matrix from this voting matrix, a transformative process is initiated. This entails the conversion of individual user voting values into binary equivalents, specifically 0 or 1, facilitated through a thresholding mechanism. A voting value of 1 is used as a threshold. This leads to a value of 1 in the binary matrix if the corresponding vote is above or equals the voting value of 1 and will be zero if the voting value is below 1 implying item is noted voted by the user. This is mathematically expressed in Eq. (1),

$$v = \begin{cases} 1 if v_{a,j} \geq 1 \\ 0 if user didn't rated j (Ø) \end{cases} \tag{1}$$

where $v_{a,j}$ denotes that user a voted value v on a certain item j. Median vote value of the dataset's voting scale or dataset's overall average value can also be used a threshold value. But it will result in information loss as an item having low vote value and an unseen item vote value both becomes zero making them non-discriminative. Tables 1 and 2 show an example user-item voting matrix and the corresponding binary matrix achieved using Eq. (1).

3.2. Bi-Clustering binary data

Clustering is a prominent technique for revealing pattern distributions and hidden correlations within extensive datasets. Notably, K-Means, a Partitional clustering approach, associates users with singular clusters based on their complete item sets. In contrast, bi-clustering enables users to belong to multiple bi-clusters simultaneously, each characterized by distinct item sets. The emergence of bi-clustering algorithms is driven by the challenge of discovering localized patterns using conventional clustering methods. These algorithms offer the unique ability to analyze all dataset dimensions concurrently, extracting intricate local patterns that enhance our understanding of underlying correlations. BiMax (Prelić et al., 2006) and Bibit (Rodriguez-Baena et al., 2011) are two binary or bi-clustering algorithms designed to work in the field of bioinformatics. In bioinformatics, the binary values 0 and 1 may correspond to gene and protein features respectively.

In this study, we employ both the BiMax and Bibit bi-clustering algorithms for the explicit purpose of enhancing product/movie recommendation. Both BiMax and Bibit algorithms are tailored to the task of bi-clustering binary matrix elements with a value of 1. BiMax adopts a divide-and-conquer approach, iteratively partitioning the binary matrix in a checkerboard pattern. Conversely, Bibit identifies patterns through the combination of any two rows within the binary matrix. Notably, both BiMax and Bibit algorithms yield dual lists for each cluster: the first list comprises indices of the rows encompassed by the cluster, while the second list comprises indices of the columns integrated within the cluster. The result of applying BiMax and Bibit algorithms on Table 2 is shown in Figs. 1 and 2. Both figures contain three colors, white corresponds to 0 s in the binary matrix, grey corresponds to 1 s in the binary matrix and black also corresponds to 1 in the binary matrix. The black 1 s are identified as a bi-cluster by the corresponding bi-clustering algorithm. On the other hand, the grey 1's doesn't belong to any bi-cluster

Table 1
Example user-item votings matrix.

	Item1	Item2	Item3	Item4
User1	5	Ø	5	1
User2	5	Ø	Ø	1
User3	4	4	5	1
User4	4	Ø	5	5
User5	1	2	Ø	5

Table 2
Binary form of Table 1.

	Item1	Item2	Item3	Item4
User1	1	0	1	1
User2	1	0	0	1
User3	1	1	1	1
User4	1	0	1	1
User5	1	1	0	1

and white 0's also doesn't belong to any bi-cluster. As the Bibit algorithm requires that we should input the size of the minimum number of rows and columns that we desire in the bi-cluster. For Table 2 we set the minimum value of rows and columns to two. Hence we can see that the identified bi-cluster contains two rows and three columns.

Fig. 1(a) shows that the BiMax algorithm identified a bi-cluster at cells {(user1, item1), (user1, item3), (user3, item1), (user3, item3)} of Table 2. To make resulting bi-cluster contiguous, rows and columns of the binary matrix (Table 2) are shuffled, thus producing a bi-cluster that is easy to view. Fig. 2(a) shows a contiguous bi-cluster formed by performing rows and columns shuffling on Fig. 1 (a). In Fig. 2(a), row2 (user 2) and row3 (user 3) of Table 2 are shuffled, and column2 (item2) and column3 (item3) are shuffled. Fig. 2(b) shows that the Bibit algorithm identified a bi-cluster at cells {(user3, item1), (user3, item2), (user3, item4), (user5, item1), (user5, item2), (user5, item4)} of Table 2. Fig. 2 (b) shows a contiguous bi-cluster formed by performing rows and columns shuffling on Fig. 2 (a). In Fig. 2(b), row3 (user 3) and row4 (user 4) are shuffled of Table 2, and column3 (item3) and column4 (item4) are shuffled.

3.3. *Measuring the quality of bi-clusters*

Numerous quality measurement functions have been proposed by researchers to quantitatively assess the quality of bi-clusters. These include Variance (VAR) (Hartigan, 1972), Mean Square Residue (MSR) (Cheng & Church, 2000), Scaling Mean Square Residue (SMSR) (Mukhopadhyay, Maulik, & Bandyopadhyay, 2009), Relevance Index (RI) (Yip, Cheung, & Ng, 2004) and correlation-based measures, such as PCC. In this work, we are using the SMSR fitness function as a quality measure of bi-clusters. SMSR is based on MSR, and the mathematical detail of MSR is given below in Eq. (2):

$$MSR(B) = \frac{1}{|I|.|J|}\sum_{i=1}^{|I|}\sum_{j=1}^{|J|}(b_{ij} - b_{iJ} - b_{Ij} + b_{IJ})^2 \tag{2}$$

where B denotes a bi-cluster consisting of I rows and J columns, b_{ij} denotes the bi-cluster value at the *ith* row and *jth* column. b_{iJ} denotes the row mean of each row i, b_{Ij} denotes the column mean of each column j and b_{IJ} denotes the overall mean of bi-cluster B. MSR measures the coherence of rows and columns in bi-cluster B. A low value of MSR indicates better quality of bi-cluster B, conversely, a high value indicates poor quality. A value of 0 indicates a perfect bi-cluster, indicating that all rows fluctuate in the same way under the experimental conditions. But ideally, such bi-clusters are hard to meet. However, MSR only capture shifting variance and not the scaling variance. So we need a mea-

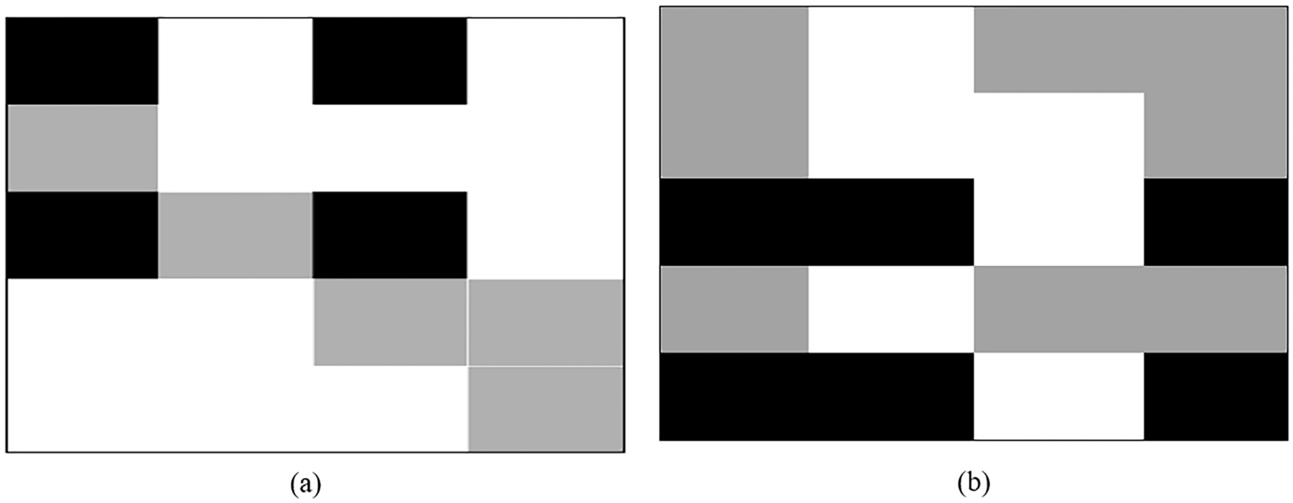


Fig. 1. **(a)** One bi-cluster created from the values in Table 2 using the BiMax algorithm **(b)** One bi-cluster created from Table 2 using the Bibit algorithm.

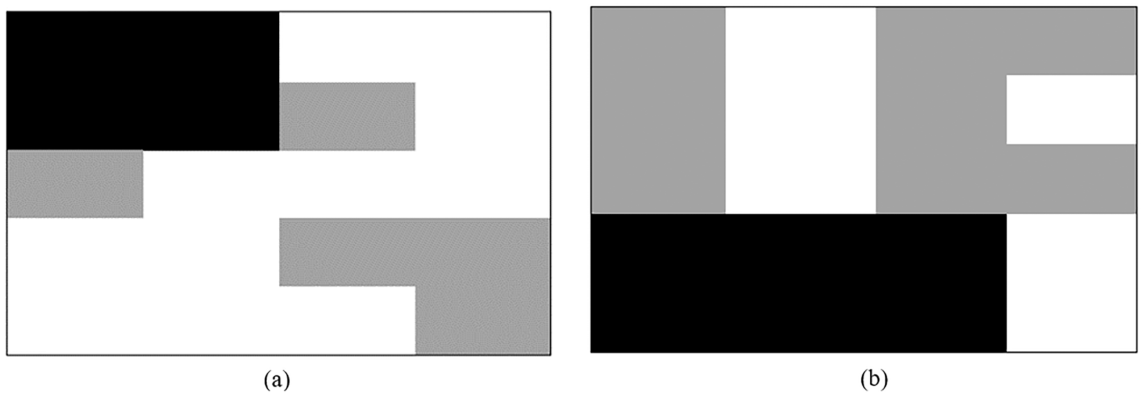


Fig. 2. Contiguous Bi-cluster formed by performing rows and columns shuffling (a) BiMax (b) Bibit.

sure that is able to capture both scaling and shifting variance. Mukhopadhyay et al. (Mukhopadhyay et al., 2009) developed SMSR to identify scaling patterns within the bi-cluster. Identification of a scaling pattern within the bi-cluster leads us to the usage of SMSR as a fitness function for the quality measure of our bi-clusters. SMSR is mathematically given in Eq. (3):

$$SMSR(B) = \frac{1}{|I|.|J|}\sum_{i=1}^{|I|}\sum_{j=1}^{|J|}\frac{(b_{iJ}\times b_{iJ} - b_{ij}\times b_{IJ})^2}{b_{iJ}^2\times b_{Ij}^2} \tag{3}$$

The values of $SMSR(B)$ are scaled in the range of 0 to 1 as compared to MSR which is not scaled from 0 to 1. So we applied SMSR on our generated bi-clusters to generate a sorted list of bi-clusters according to their quality values.

3.4. Placement of bi-clusters in a square grid

Following the generation and subsequent sorting of bi-clusters based on their SMSR values, the subsequent step involves establishing a RL environment to facilitate our recommendation agent in generating personalized recommendations. Our RL environment consists of a Square Grid (SG) of size $n \times n$. The placement of sorted bi-clusters onto the designated Square Grid (SG) presents a pivotal task. Yet, this endeavor is not without its challenges, encompassing three notable challenges.

We shall.

- **Challenge#1:** What if the number of bi-clusters are greater than the size of SG? For example, if the size of the SG is $3 \times 3 = 9$ and the number of bi-clusters are greater than 9. To solve this, we propose
- bi-cluster merging.
- **Challenge#2:** What if the number of bi-clusters is smaller than the size of SG? For example, if the size of the square grid is $3 \times 3 = 9$ and the number of bi-clusters is smaller than 9. To solve this, we propose a decomposition of the poor-quality bi-clusters using the same bi-clustering algorithm that is used to generate bi-clusters.
- **Challenge#3:**What should be the placement of bi-clusters in the *SG*? The placement of bi-clusters in the *SG* should follow two principles:
  - o **Principle#1:** Bi-clusters having closed SMSR values should be close to each other on the *SG*.
  - o **Principle#2:** Bi-clusters having a large difference in SMSR values should be furthest from each other on SG.

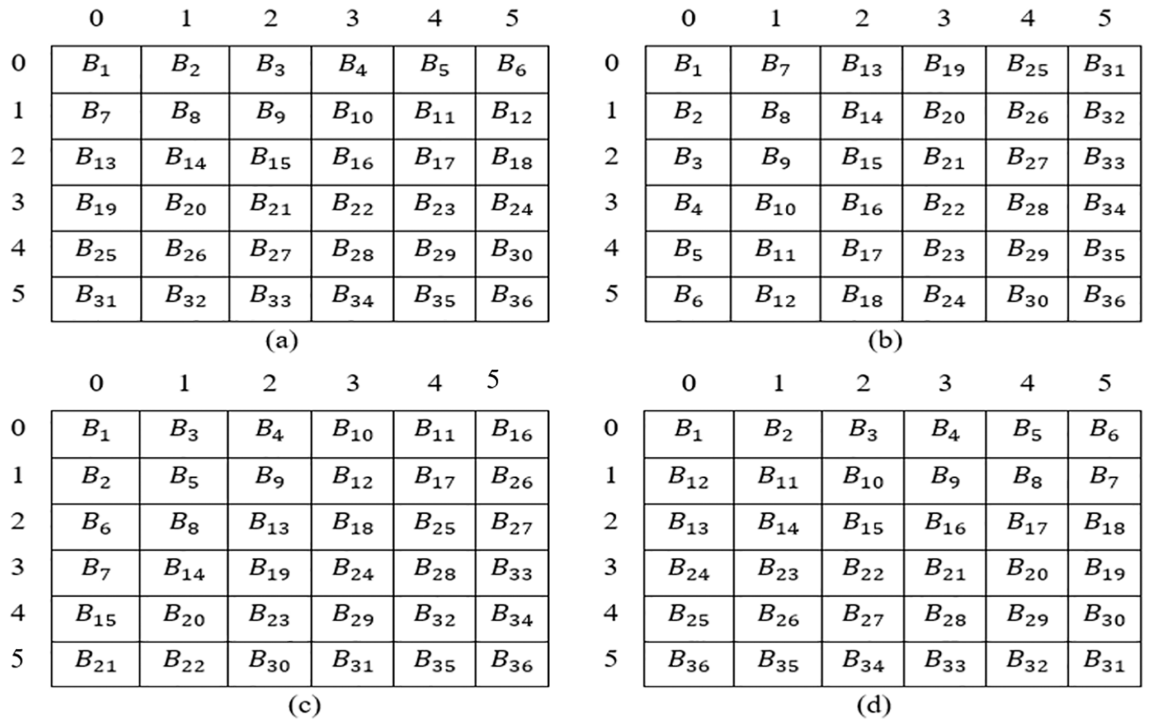
Space-filling curves provide us with the base to satisfy the two principles stated above. Space-filling curves are curves that can visit all possible points in the multidimensional space, in our case the $n \times n$ square grid. Suppose we have bi-clusters in the range of 1 to 36 $= \{B_1, B_2, B_3, \cdots..B_{36}\}$and the grid size is 6×6. Four methods are given by Candan and Sapino (Candan & Sapino, 2010) to position one-dimensional data on a two-dimensional grid. These methods include row order traversal, column order traversal, cantor diagonal and row prime order traversal. How these methods will place different bi-clusters in each grid cell is illustrated in Fig. 3. Placement of data on a grid having dimension $n \times n$ is given by Eq. (4):

$$C_{\pi\_order}(\overrightarrow{v}) = \sum_{i=1}^{m}\overrightarrow{v}[\pi(i)] \times (2^n)^{m-i} \tag{4}$$

For a 2D grid, $\pi(1)$ corresponds to the first dimension and $\pi(2)$ corresponds to the second dimension. In this way, we can find cell $(1,0)$ in the 6×6 grid that will contain which bi-cluster in row-order traversing. For example, $C_{roworder}(1,0) = 1 \times 6^1 + 0 \times 6^0 = 6 = B_7$. While in column order traversal cell $(1,0)$ will contain B_2 and in cantor-diagonal traversal grid cell $(1,0)$ will also contain B_2. One common thing in all three types of traversals is that the best bi-cluster (bi-cluster with the lowest SMSR value) is at the first position and the worst bi-cluster is at the last position. More complex grid positions like the Hilbert curve and Z-order traversal are also available but they will make our method more computationally intensive.

3.5. Bi-cluster merging

To adapt bi-clusters to the dimensions of the squared grid, a process



(a)

	0	1	2	3	4	5
0	B_1	B_2	B_3	B_4	B_5	B_6
1	B_7	B_8	B_9	B_{10}	B_{11}	B_{12}
2	B_{13}	B_{14}	B_{15}	B_{16}	B_{17}	B_{18}
3	B_{19}	B_{20}	B_{21}	B_{22}	B_{23}	B_{24}
4	B_{25}	B_{26}	B_{27}	B_{28}	B_{29}	B_{30}
5	B_{31}	B_{32}	B_{33}	B_{34}	B_{35}	B_{36}

(b)

	0	1	2	3	4	5
0	B_1	B_7	B_{13}	B_{19}	B_{25}	B_{31}
1	B_2	B_8	B_{14}	B_{20}	B_{26}	B_{32}
2	B_3	B_9	B_{15}	B_{21}	B_{27}	B_{33}
3	B_4	B_{10}	B_{16}	B_{22}	B_{28}	B_{34}
4	B_5	B_{11}	B_{17}	B_{23}	B_{29}	B_{35}
5	B_6	B_{12}	B_{18}	B_{24}	B_{30}	B_{36}

(c)

	0	1	2	3	4	5
0	B_1	B_3	B_4	B_{10}	B_{11}	B_{16}
1	B_2	B_5	B_9	B_{12}	B_{17}	B_{26}
2	B_6	B_8	B_{13}	B_{18}	B_{25}	B_{27}
3	B_7	B_{14}	B_{19}	B_{24}	B_{28}	B_{33}
4	B_{15}	B_{20}	B_{23}	B_{29}	B_{32}	B_{34}
5	B_{21}	B_{22}	B_{30}	B_{31}	B_{35}	B_{36}

(d)

	0	1	2	3	4	5
0	B_1	B_2	B_3	B_4	B_5	B_6
1	B_{12}	B_{11}	B_{10}	B_9	B_8	B_7
2	B_{13}	B_{14}	B_{15}	B_{16}	B_{17}	B_{18}
3	B_{24}	B_{23}	B_{22}	B_{21}	B_{20}	B_{19}
4	B_{25}	B_{26}	B_{27}	B_{28}	B_{29}	B_{30}
5	B_{36}	B_{35}	B_{34}	B_{33}	B_{32}	B_{31}

Fig. 3. (a) Row order traversal of 6×6 grid. (b) Column order traversal of 6×6 grid. (c) Cantor-diagonal traversal of 6×6 grid. (d) Row prime order traversal of 6×6 grid.

of merging bi-clusters was employed, specifically targeting those bi-clusters with lower SMSR quality function values. Such bi-clusters of inferior quality were consolidated with higher-quality counterparts. To achieve this, an analysis was conducted to assess the extent of overlap and similarity between the less optimal bi-clusters and their more favorable counterparts. This assessment encompassed evaluating the degree of user and item overlap within both the lower-quality and higher-quality bi-clusters.. The similarity of both bi-clusters is measured using ITR similarity (Iftikhar et al., 2020). A detailed description of ITR similarity is given in section 3.6. ITR similarity first computes triangle $'$ similarity between column-wise average voting vectors of both bi-clusters and then computes the User Ratings Preferences (URP) between both bi-clusters. To obtain the average and standard deviation values required for URP, column-wise average voting vectors of both bi-clusters are computed and then the average voting value and the standard deviation is computed for this average voting vector. Table 3 presents different cases of bi-cluster merging.

Suppose we have two bi-clusters, Bi-cluster1 and Bi-cluster 2, as shown in Fig. 4, and we want to perform a merging of these two bi-clusters. Fig. 5 pictorially show four merging cases for these two bi-clusters.

3.5.1. Example of bi-cluster merging

Fig. 6 presents an example of bi-cluster merging for bi-clusters 1 and 2. Where, rows of merged bi-cluster corresponds to union set of users of both bi-clusters 1 and 2 and columns corresponds to union set of items of both bi-clusters 1 and 2.

- Bi-cluster1 Users={U_4, U_6, U_7}
- Bi-cluster2 Users={U_6, U_7, U_8}
- Bi-cluster1 Items={I_1, I_2, I_3}
- Bi-cluster2 Items={I_3, I_4, I_5}
- Users overlapping in both Bi-clusters ={ U_6, U_7}
- Items overlapping in both Bi-clusters ={ I_3}
- Union set of users in both bi-clusters={U_4, U_6, U_7, U_8}
- Item set of items in both bi-clusters={I_1, I_2, I_3, I_4, I_5}
- Merged Bi-cluster Users={U_4, U_6, U_7, U_8}
- Merged Bi-cluster Items={I_1, I_2, I_3, I_4, I_5}

In order to obtain a merged bicluster we used a very simple technique. We make union set of users and items in both bi-clusters 1 and 2. As these union set of users and items are subset of user and item set of actual user-item voting database and rows of merged bi-cluster should correspond to union set of users and columns of merged bi-cluster should correspond to union set of items. So, to generate a merged bi-cluster we extracted each user and item corresponding value from actual user-item voting database.

3.6. MDP notation

In an MDP, we have a set of states S, a set of actions A, and a set of rewards R. The state space $S = \{B_1, B_2, B_3, B_4 \cdots B_n, \}$ and the action space $A = \{Up, Left, Down, Right, \}$. We shall assume that each of these sets has a finite number of elements. At each time step $t = 0, 1, 2, \cdots$, the agent receives some representation of the environment's state $S_t \in$ S. Based on this state, the agent selects an action $A_t \in$ A, this gives us the state-action pair (S_t, A_t). Time is then incremented to the next time step $t + 1$, and the environment is transitioned to a new state $S_{t+1} \in$ S. At this time, the agent receives a numerical reward $R_{t+1} \in$ R for the action A_t taken from state S_t. Thus, the reward can be thought of as an arbitrary function f that maps the state-action pairs to rewards. Thus at each time step t we have $f(S_t, A_t) = R_{t+1}$. A trajectory showing the sequential process of selecting an action from a state, transitioning to a new state, and receiving a reward can be represented as $S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots$

Table 3
Different possible cases for merging bi-clusters.

Cases	Overlapping	Similarity	Merging
Case 1	No	No	No
Case 2	Yes (But only row wise)	Less than a threshold	No
Case 3	Yes (But only column wise)	Less than a threshold	No
Case 4	Yes	Greater than a threshold	Yes

3.6.1. Transition probabilities

As both sets S and R are finite, the random variables S_t and R_t should have well-defined probability distributions. That is, all possible values of S_t and R_t should have some associated probability and depends upon the previous state only. For all $s' \in$ S, $s \in$ S, r $\in$ R and $a \in$ A(s), we define the probability of the transitioning to state s' with reward R from taking action a in state s as given in Eq (5):

$$P(s', r|s, a) = \Pr\{S_t = s', R_t = R | S_{t-1} = s, A_{t-1} = a\} \tag{5}$$

Start state: Deciding the start state is important as subsequent recommendations heavily rely on the start state. Choi et al. (Choi et al., 2018) use Jaccard similarity to decide the start state. However, Jaccard ignores the true value of the voting and also may have the same value for many bi-clusters. Sargar (Sargar, 2020) used cosine similarity instead of Jaccard similarity but the obvious drawback of cosine similarity is that it only considers angles between voting vectors and ignores the length of the voting vector. We decided on the start state by computing the ITR similarity between the target user voting vector and the average voting vector of each bi-cluster (Iftikhar et al., 2020). The reason to use ITR similarity is that it utilizes the complete rating vector of both the target user and bi-cluster and also considers global perspectives like average values and standard deviation.

$$sim^{TRIANGLE'}_{(u,bic)} = 1 - \frac{\sqrt{\sum_{t\in T}(v_{u,t} - v_{bic,t})^2}}{\sqrt{\sum_{t\in T} v_{u,t}^2} + \sqrt{\sum_{t\in T} v_{bic,t}^2}} \tag{6}$$

$$sim^{urp}_{(u,bic)} = 1 - \frac{1}{1 + exp(-|\bar{v}_u - \bar{v}_{bic}| . |\sigma_u - \sigma_{bic}|)} \tag{7}$$

where $\bar{v}_u$ is the average voting of target user u, $\bar{v}_{bic}$ is the average voting of a bi-cluster, σ_u is the standard deviation of the target user u, and σ_{bic} is the standard deviation of a bi-cluster.

$$ITRsim_{(u,bic)} = sim^{TRIANGLE'}_{(u,bic)} \times sim^{urp}_{(u,bic)} \tag{8}$$

Pit/Hole state: If an agent reaches to a state that has no similarity with neighbor states, then that state is termed a pit/hole state. As the agent is unable to move anywhere from this state.

Goal state: Agent getting off the grid, not getting any new recommendations or coming to the start state after the move.

Number of Episodes: The number of episodes is fixed to 100.

The maximum number of steps in each episode: The number of steps in each episode is constrained by the size of the grid. For our grid size having 36 states and 4 actions in action space, this number will be the number of states multiplied by the number of actions in the action space resulting in 144, the number of steps.

Maximizing Return: In MDP, the agent's goal is to maximize the expected discounted return of rewards (G_t) given in Eq. (9). Here, γ is a discount factor representing that an immediate reward is better than an equal-valued future reward. Thus, γ value decreases for each increase in time step.

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} \cdots$$

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

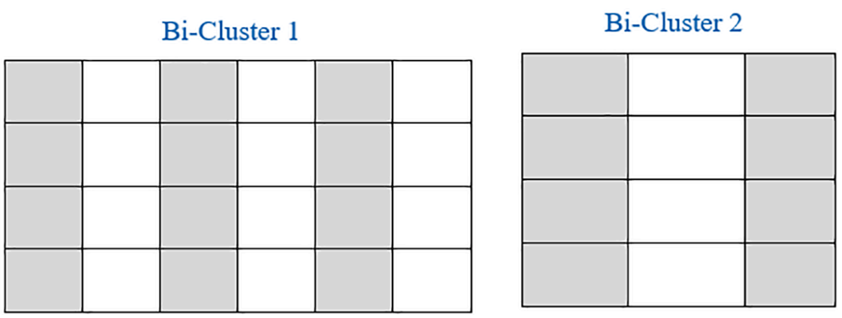
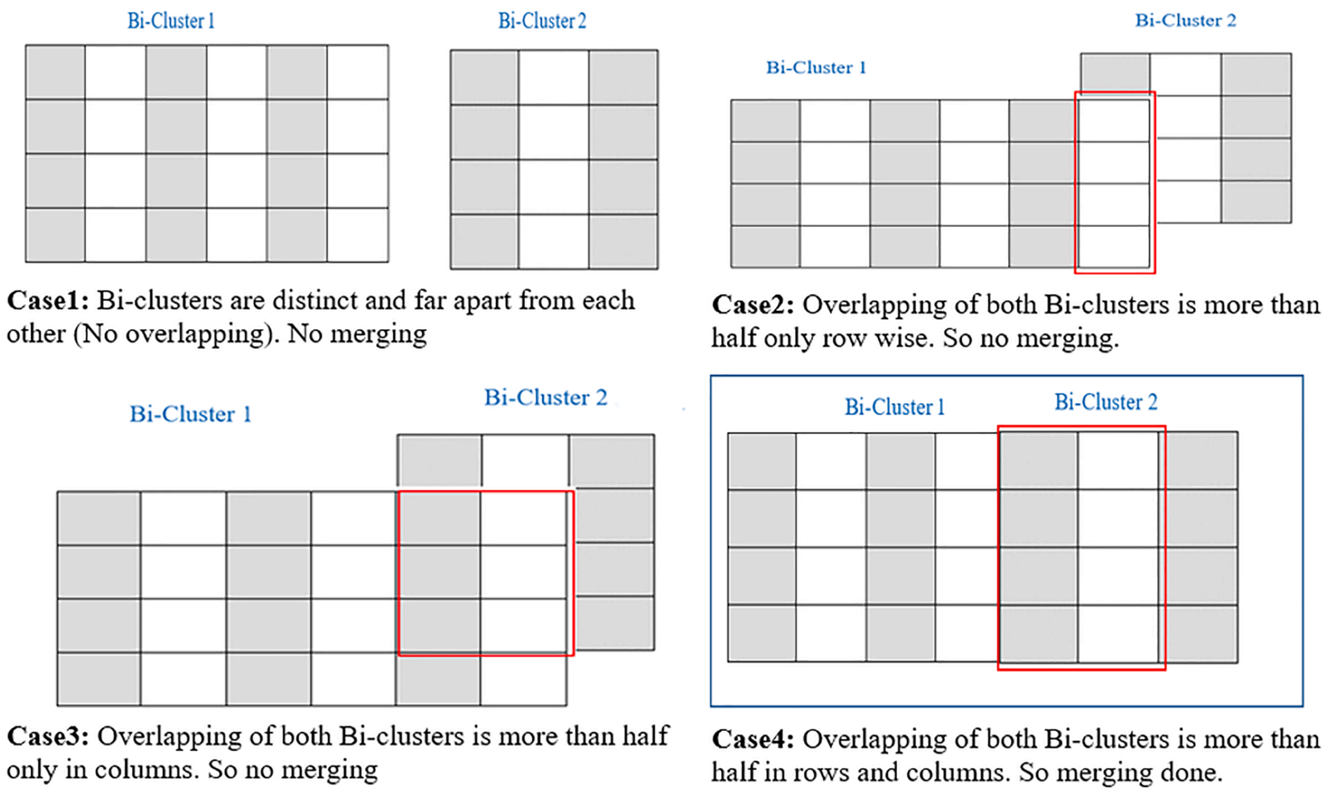


Fig. 4. Two candidate bi-clusters for merging.



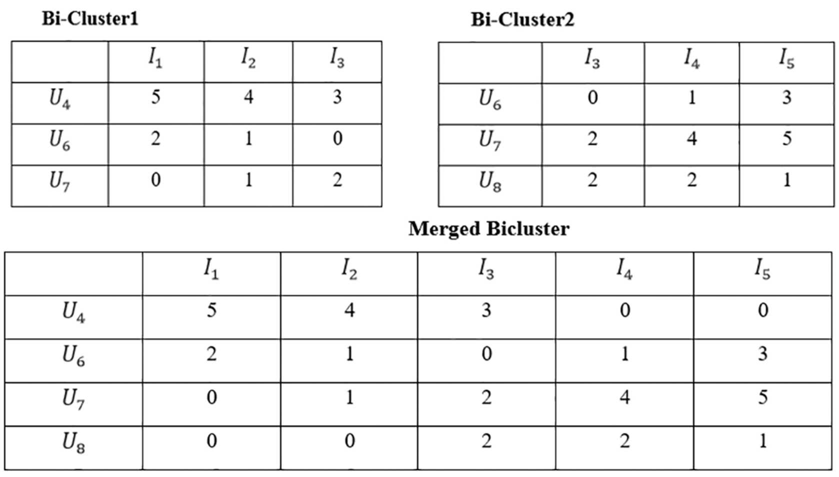
Case1: Bi-clusters are distinct and far apart from each other (No overlapping). No merging

Case2: Overlapping of both Bi-clusters is more than half only row wise. So no merging.

Case3: Overlapping of both Bi-clusters is more than half only in columns. So no merging

Case4: Overlapping of both Bi-clusters is more than half in rows and columns. So merging done.

Fig. 5. Illustration of four possible cases of bi-clusters merging.



Bi-Cluster1

	I_1	I_2	I_3
U_4	5	4	3
U_6	2	1	0
U_7	0	1	2

Bi-Cluster2

	I_3	I_4	I_5
U_6	0	1	3
U_7	2	4	5
U_8	2	2	1

Merged Bicluster

	I_1	I_2	I_3	I_4	I_5
U_4	5	4	3	0	0
U_6	2	1	0	1	3
U_7	0	1	2	4	5
U_8	0	0	2	2	1

Fig. 6. Bi-cluster merging example.

$$RU_{t+1} = \frac{|U_{s_t} \cap U_{s_{t+1}}|}{|U_{s_t} \cup U_{s_{t+1}}|}$$

$$RI_{t+1} = \frac{|U_{s_t} \cap U_{s_{t+1}}|}{|U_{s_t} \cup U_{s_{t+1}}|}$$

$$R_{t+1} = RU_{t+1} + RI_{t+1} \tag{9}$$

A reward (R_{t+1}) is computed using Jaccard similarity. Reward (RU_{t+1}) depends on the number of overlapping users in states s_t and s_{t+1} and reward (RI_{t+1}) depends on the number of overlapping items in states s_t and s_{t+1}. More overlapping in both states results in a better reward value, minor overlapping gives a low reward and no overlapping results in zero rewards. If an agent chooses an action that leads to the same current state, then the reward will be zero. Both types of rewards ($RU_{t+1} and RI_{t+1}$) are then combined into a single-valued reward (R_{t+1}). This overcomes the inherent drawback of CF, which at any one time can use only user-based CF or item-based CF. Finally, the agent's objective is to move to a state that gives better reward value which in turn will result in maximum return G_t. Our proposed methodology is given in Fig. 7.

Optimal policy $\pi^*(s)$ learning: In RL, we define a policy $\pi(s)$ that tells us to perform which action $a \in$ A in a particular state $s \in$ S. RL helps us to find the optimal policy $\pi^*(s)$, that maximizes the expected return G_t. We shall learn the optimal policy $\pi^*(s)$ by a state-action value function $Q_\pi(s,a)$, which indicates that the expected value of the return G_t obtained from episodes starting from a certain state s can be expressed as Eq. (10):

$$Q_\pi(s,a) = E_\pi\{G_t|s_t = s, a_t = a\} = E_\pi\{\sum_{k=0}^{\infty} \gamma^k R_{t+k}|s_t = s, a_t = a\} \tag{10}$$

At the start, the agent has no idea of the environment, so it starts an exploration of the environment. As soon as the agent's learning of the environment (exploration) increases, the agent starts exploiting the environment. Exploration is a dilemma in which an agent to improve its current knowledge about each action in such a way that increases his long-term benefit. Improving knowledge in this way increases the accuracy of the estimated action-values for agent thus enabling the agent to take more informed decisions at some later time. Exploitation is a dilemma in which agent chooses the greedy action to get the most reward by exploiting the agent's current action-value estimates. But by being greedy in terms of action-value estimates, may result in reduced reward, thus resulting in a sub-optimal behavior of the agent. When an agent explores, it gets more accurate estimates of action-values. And when it exploits, it might get more reward. It cannot, however, choose to do both simultaneously, which is also called the exploration–exploitation dilemma. To balance exploration and exploitation, we used $\in$ -greedy method (Sutton & Barto, 1998b). $\in$ -Greedy is a simple method to balance exploration and exploitation by choosing between exploration and exploitation randomly. The $\in$ -greedy, where $\in$ denotes to the probability of choosing to explore, exploits most of the time with a small chance of exploring. Mathematical working of $\in$ -greedy algorithm is given in Eq. (11).

$$Actionattime(t) = \begin{cases} \max Q_t(a) withprobability 1 - \in \\ anyactionawithprobability \in \end{cases} \tag{11}$$

To learn optimal policy, Q-learning, is used. Q-learning is an off-policy approach, which aim to determine best action from current state and thus accomplishing it's own set of rules to determine optimal policy. Internally, Q-learning, builds a Q-table containing all possible states and actions as shown in Table 4. Initially, values of all actions in a state are set to zero. Then agent start it's movement while balancing exploration and exploitation and updates value of each action in Q-table using Eq. (12).

$$Q(s,a) = Q(s,a) + \alpha^*(R + \Upsilon^* \max(Q(s',a')) - Q(s,a)) \tag{12}$$

In Eq. (12) $Q(s,a)$ represents the expected reward for taking

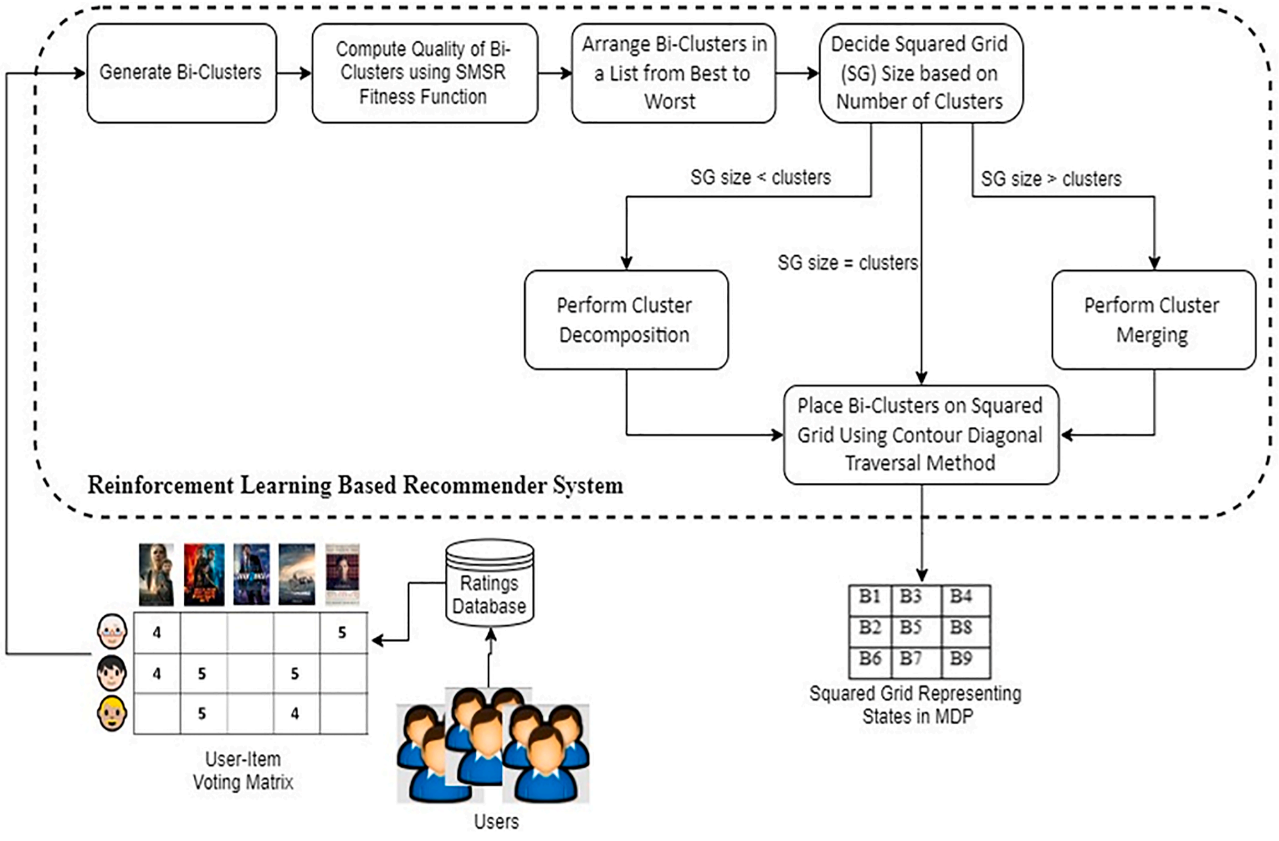

Generate Bi-Clusters
Compute Quality of Bi-Clusters using SMSR Fitness Function
Arrange Bi-Clusters in a List from Best to Worst
Decide Squared Grid (SG) Size based on Number of Clusters
SG size < clusters
SG size > clusters
SG size = clusters
Perform Cluster Decomposition
Perform Cluster Merging
Place Bi-Clusters on Squared Grid Using Contour Diagonal Traversal Method
Reinforcement Learning Based Recommender System
Ratings Database
User-Item Voting Matrix
Users
B1 B3 B4
B2 B5 B8
B6 B7 B9
Squared Grid Representing States in MDP

Fig. 7. Flow diagram of the proposed methodology.

Table 4
Snapshot of Q-Table at time t_0 for Q-Learning.

	Actions			
States	**Left**	**Right**	**Up**	**Down**
B_1	0	0	0	0
B_2	0	0	0	0
B_3	0	0	0	0
B_4	0	0	0	0
….	0	0	0	0
B_n	0	0	0	0

action a in state s. The actual reward received for that action is referenced by R while s' refers to the next state. The learning rate is α and γ is the discount factor.The highest expected reward for all possible actions a $'$ in state s' is represented by $max(Q(s', a'))$. Learning rate α gives us the probility of how value of $Q(s, a)$ will be updated in Eq. (12). If α is zero then $Q(s, a)$ will be updated based upon past experience and if α is one then $Q(s, a)$ will be updated based upon current situation. Discount factor γ gives us the option how reward will be accounted towards policy selection. If γ is zero then only current rewards are accounted and if is γ one then only future rewards will also be taken into account. Eq. (12) is used to update value of each action for each state in Q-table (Table 5).

Table 5 shows an insight of different action probabilities (Q values) for each state at some time t_0. This table is build and updated by the agent while visiting each state and balancing exploration vs exploitation dilemma as given in Eq. (11). Initially, agents explore more and exploits less. As soon as this table going to build on, agent's exploitation increases and exploration decreases. Eq. (11) gives the agent a probability to choose which action on given state. Q value at that given state and action is updated using Eq. (12). Agent performs a number of episodes to build and update this table. In Table 5, suppose the environment is deterministic and suppose determined start state (using Eq. (8)) of agent is B_1 then agent will take the action with the highest probability, 0.902 (Right), among all other action probabilities and will land in to state B_3 of Fig. 3 (c). In state B_3 the agent will take action *down*, having a probability of 0.535, and will land in the next state B_5. In this way, the agent will continue its move until it reaches a goal state (having no change in the set of recommended items) or to a pit/hole state.

Q-learning to solve a reinforcement learning problem is considered as an off-policy method. In off-policy methods, RL agent learns the value of $Q(s, a)$ using the actions derived from another policy. Another, good alternative to Q-learning to solve reinforcement learning problem is State-Action-Reward-State-Action(SARSA). SARSA is an on-policy method as in on-policy methods, RL agent learns the value $Q(s, a)$ by the actions derived from the current policy. Value update rule is slightly different from Q-learning and is given in Eq. (13).

$$Q(s, a) = Q(s, a) + \alpha^*(R + \Upsilon^* Q(s', a')) - Q(s, a)) \tag{13}$$

where $Q(s, a)$ represents the expected reward for taking action a in state s. The actual reward received for that action is referenced by R while s' refers to the next state and a' refers to the next action. Internally, SARSA also uses same Q-table for optimal policy learning.

Table 5
Snapshot of Q-Table at a later time for Q-Learning.

	Actions			
States	**Left**	**Right**	**Up**	**Down**
B_1	0.735	0.902	0.663	0.814
B_2	0.625	0.125	0.25	0.333
B_3	0	0	0	0.535
B_4	0.485	0.628	0.663	0.714
B_5	0.175	0.345	0.082	0.05
….	….	….	….	….
B_{36}	0.284	0.045	0.764	0.902

4. Experimentation procedure

Detailed experimentation procedure is given in this section.

4.1. Experimentation and dataset description

We used the Movie Lens ML-100 k and FilmTrust datasets for experimentation purposes. ML-100 k dataset is comprised of 100,000 votes given by 943 users on 1,682 distinct movies. The input space becomes too large for the RL agent to work upon if we directly apply it to the original user-item voting matrix, containing 943 rows and 1682 columns, therefore we applied bi-clustering on our matrix to reduce the input dataset dimensions. Among many bi-clustering algorithms, we have used BiMax (Prelić et al., 2006) and Bibit (Rodriguez-Baena et al., 2011), the most famous. The Bibit (Rodriguez-Baena et al., 2011) algorithm takes two input parameters, the minimum number of rows (*mnr*) and minimum number of columns (*mnc*). Initially, the value of *mnr* is set to 4 and *mnc* is set to 3 (as per (Rodriguez-Baena et al., 2011)) and the algorithm was applied to the ML-100 k dataset. This resulted in 197,231 bi-clusters, a very large number. Through experimental investigation of these parameters, we fine-tuned the values of *mnr* and *mnc* to 70 and 10, respectively, which resulted in 40 bi-clusters. The proposed grid size was set to 6 × 6 and for that bi-cluster merging (as proposed in Section 3.5) was applied to fit the grid size, resulting in 36 bi-clusters. Bi-clusters generated using BiMax were also merged to fit the size of the grid. These 36 bi-clusters contain 28,901 movie votings only. Below is the placement of sorted bi-clusters on the 6 × 6 grid using cantor-diagonal traversal method, as presented in Fig. 3(c).

B_0	B_1	B_5	B_7	B_{13}	B_{17}
B_2	B_4	B_6	B_{15}	B_{14}	B_{27}
B_3	B_8	B_9	B_{21}	B_{16}	B_{32}
B_{10}	B_{11}	B_{20}	B_{22}	B_{31}	B_{29}
B_{12}	B_{19}	B_{24}	B_{25}	B_{26}	B_{30}
B_{18}	B_{23}	B_{33}	B_{28}	B_{34}	B_{36}

The FilmTrust dataset is comprised of 35,497 votes given by 1,508 users on 2,071 distinct movies. Again, for the Bibit bi-clustering algorithm (Rodriguez-Baena et al., 2011), through experimental investigation, the values of *mnr* and *mnc* were fine-tuned to 110 and 49, respectively, resulting in 41 bi-clusters. The proposed grid size was set to 6 × 6 and bi-cluster merging was applied to fit the bi-clusters to the grid size, resulting in 36 bi-clusters. Bi-clusters generated using BiMax were also merged to fit the size of the grid. Below is the placement of the sorted bi-clusters on a 6 × 6 grid using the cantor-diagonal traversal method, as presented in Fig. 3(c) for the FilmTrust dataset.

B_0	B_1	B_5	B_6	B_{14}	B_{17}
B_2	B_4	B_8	B_{15}	B_{16}	B_{26}
B_3	B_7	B_{12}	B_{13}	B_{21}	B_{27}
B_9	B_{11}	B_{19}	B_{25}	B_{23}	B_{38}
B_{10}	B_{18}	B_{24}	B_{36}	B_{31}	B_{32}
B_{20}	B_{22}	B_{34}	B_{30}	B_{29}	B_{28}

All code was written in Python using Pycharm community edition as IDE and executed on a core-i5 Dell laptop with 8-GB RAM. For the 2D grid environment creation, OpenAI gym is used.

4.2. Methods used for comparison

We compared our proposed method with the following RL and non-RL methods:

- **RLRS1 (Sargar, 2020):** Sargar applied RL on bi-clusters placed on an $n \times n$ square grid, using a filtering mechanism to filter out good-quality bi-clusters, leaving out poor-quality clusters. Leaving poor-quality bi-clusters results in information loss and affects recommendation accuracy. The start state was determined using Cosine similarity. The reward was based only on the overlapping of users in both bi-clusters.

- **IPWR (Pure CF method) (**Ayub et al., 2019**):** Ayub et al. used this method to recommend items to users. The similarity was computed by improving the PCC measure to include both user and item averages. The improved PCC was then combined with the user's RPB.
- **ITR (Pure CF method) (**Iftikhar et al., 2020**):** In this method from Iftikhar et al., the similarity of two users are computed by representing user voting vectors in the form of a triangle. The computed triangle similarity is combined with the voting preferences of users (URP).
- **S1 (Partitional Clustering method) (**Zahra et al., 2015**):** In this method twenty-one different methods were suggested to choose the initial centroids of clusters by manipulating the underlying data correlation structure. We only selected method 1 out of twenty-one methods, namely S1 of this research paper. The S1 method chooses k users uniformly at random as initial centroids.
- **SPOP (Partitional Clustering method) (**Khalid et al., 2017**):** Khalid et al., assigned users to different hyperspheres by computing the distance from the centre of the hypersphere. The distance measuring approach used was Pearson correlation with default votes. Assuming default votes for non-voted items is just an approximation and not a good choice. As users may not be willing to vote for items whose vote values are approximated.

4.3. Evaluation metrics

In this section, we will discuss the evaluation metrics chosen to assess our proposed method, selected based on their common usage in evaluating recommendation algorithms. (Silveira et al., 2019).

- **Latency:** The time taken by the algorithm to generate suggestions/recommendations.
- **The return earned in each episode:** The agent performs a fixed number of 100 episodes to reach the goal state. Some of these episodes may be successful and some may be unsuccessful. In either case, the agent earns some positive reward or no reward. These rewards are accumulated for each episode and show how quickly or slowly the agent is learning from its experience while interacting with the environment.
- **The number of steps taken in each episode:** By starting the movement from the start state, the agent wants to reach a goal state. The number of steps taken by the agent to reach the goal state in each episode is recorded. The objective is to reach the goal state by taking fewer steps.
- **∈-greedy value to balance exploration and exploitation:** At the start, the agent has no idea of the environment, so the ∈ -greedy algorithm is used to balance exploration and exploitation. The ∈ -greedy start value is set to 0.9 and the final value is set to 0.1, with a decay of 0.999. A start value of 0.9 indicates that at the start, the agent explores the environment and as the agent's learning increase with the execution of each episode, exploitation is increased and exploration is decreased.
- **Precision:** Precision is measured as the percentage of overlapped items/movies of test users and items/movies recommended by the proposed method to the total set of items/movies recommended by the proposed method.

$$Precision = \frac{|I_T \cap I_P|}{|I_P|} \tag{14}$$

- **Recall:** Recall is measured as percentage of overlapped items/movies of test users and items/movies recommended by the proposed method to the total set of items/movies available in the test set.

$$Recall = \frac{|I_T \cap I_P|}{|I_T|} \tag{15}$$

- **F-measure:** F-measure, also known as the F1 score, is the harmonic mean of both precision and recall. A high value of F-measure indicates better model response and performance.

$$Fmeasure = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{16}$$

- **Item Coverage:** Indicates the extent to which items/movies in test set are accurately recommended by the proposed method.

$$ItemCoverage = \frac{|I_T|}{|I_P|} \tag{17}$$

5. Results & analysis

This section is divided into two subsections: Section 5.1 discusses results from the ML-100 K dataset, and Section 5.2 presents results from the FilmTrust dataset.. The value of learning rate α is set to 0.5 and value of discount factor γ is set to 0.95. Reason for 0.5 value of α is that we want to give equal importance to both current and past experience in Eq. (12), (13). Value of discount factor γ is set to 0.95 implying we are giving 95% importance to future rewards for updating value of $Q(s, a)$ in Eq. (12), (13), and only 05% importance to current rewards.

5.1. Latency

A random user was selected, and policy learning, policy extraction, and recommendation generation times were recorded. The algorithm was executed 10 times, and the average outcomes are reported. In the policy learning phase, the algorithm was iterated for 100 episodes, capturing the return acquired in each episode within the Q-table. Subsequently, during policy extraction, the Q-table was scrutinized to extract the optimal policy.. In applying the policy step, the learned optimal policy is applied to generate recommendations. We observed the following for the ML-100 K dataset:

- Average policy learning time for a single user: 0.05 sec
- Average policy extraction time for a single user: 0.01 sec
- Average applying policy time and getting recommendation for a single user: 0.03

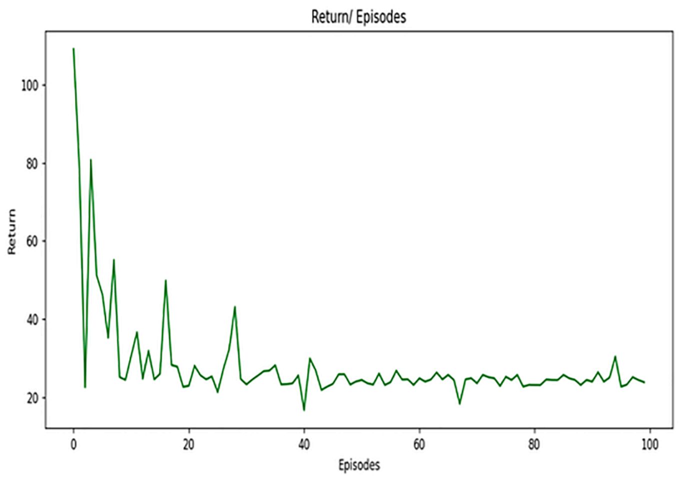
We observed the following for the FilmTrust dataset:

- Average policy learning time for a single user: 0.07 sec
- Average policy extraction time for a single user: 0.01 sec
- Average applying policy time and getting recommendation for a single user: 0.03

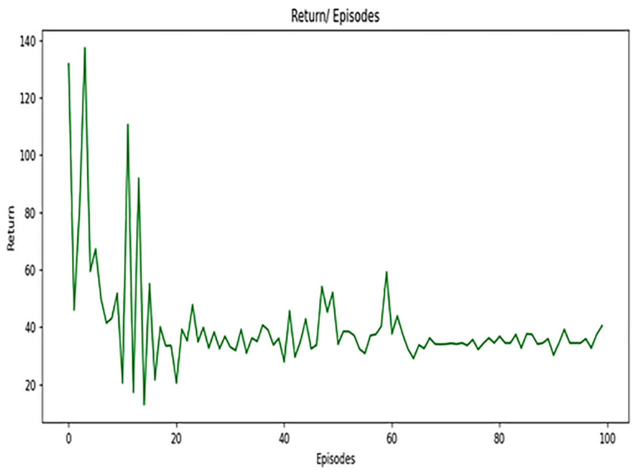
Policy extraction and policy application times exhibit comparable durations across both datasets. However, the policy learning time diverges between the two datasets. This variance can be attributed to the differential sizes of the bi-clusters within the FilmTrust dataset, which are notably larger compared to those in the ML-100 K dataset.

5.2. Return earned in each episode

In the context of the ML-100 K dataset, distinct returns were acquired by the agent in each episode, as illustrated in Fig. 8(a). This return ranges from 20 to 120 for different episodes. A low return indicates that the agent's movement is terminated in the early stages. A high return value indicates that the agent moved to many different states before reaching the goal state. But on average we can observe that the agent is



(a) ML-100K dataset



(b) FilmTrust dataset

Fig. 8. Earned return in each episode, over 100 episodes for a single user on each dataset.

receiving 20 to 40 rewards per episode. From episodes 40 to 100, the lowest fluctuation in reward is observed.

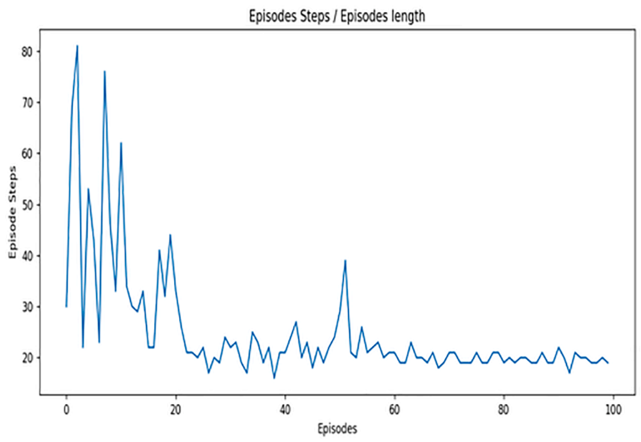
In the context of the FilmTrust dataset, the agent garnered diverse returns for each episode, evident from Fig. 8(b). This spectrum of returns spans from 5 to 140 across varying episodes. On average, the agent seems to receive rewards ranging between 20 and 50 per episode. Notably, episodes 60 to 100 exhibit the least volatility in reward fluctuations.

5.3. Number of steps taken in each episode

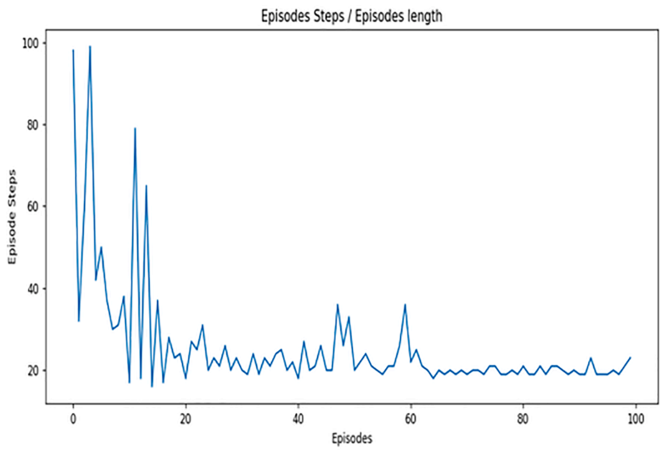
Fig. 9(a) reveals a notable trend within the ML-100 K dataset, where a substantial portion of episodes encompassed approximately fifty steps. This signifies that the agent navigated through around fifty bi-clusters before reaching the goal state. Episodes exhibited a minimum of ten steps and a maximum of 80 steps. Particularly, in the initial episodes spanning from 1 to 15, the occurrence of maximum steps is evident. When juxtaposed with Fig. 11, a distinct pattern emerges, indicating that the agent predominantly visits a higher number of states at the outset. As a consequence, it can be deduced that both the return and the number of steps to access diverse states are notably higher in the early stages.. Analyzing Fig. 9(b) for the FilmTrust dataset, we observe that the majority of episodes span approximately thirty steps, signifying the exploration of thirty bi-clusters prior to reaching the goal state. Episodes range from a minimum of ten to a maximum of 140 steps. Initial episodes, 1 to 18, exhibit maximum steps. Notably, episodes 60 to 100 show a reduced minimum step count, indicating intensified initial state exploration, subsequently leading to decreased return and step counts as agent learning progresses.

5.4. ∈ -greedy value to balance exploration and exploitation

Within the context of the ML-100 K dataset, Fig. 10(a) aptly portrays the agent's progressive grasp of the environment, aligning with the rise in episode count. The initial ∈ -greedy value is set to 0.9, indicating that at the start, the agent will explore more. The final ∈ -greedy value is set to 0.1 with a decay rate of 0.099. A decaying curve indicates the agent's learning. At the start, when the agent has no idea of the environment it will explore the environment, but in subsequent episodes, when the agent becomes familiar with the environment, it will start exploiting the environment. Fig. 10(b) shows that the agent's learning of the environment is increasing with the increase in the number of episodes. Initial ∈ -greedy value is set to 0.9 indicating that in start agent will explore more. Final ∈ -greedy value is set to 0.1 with the decay rate of 0.099. A decaying curve indicates agent's learning. In its initial stages, the agent, driven by limited environmental awareness, engages in exploratory actions. Subsequent episodes witness a transition toward exploitation as the agent's comprehension evolves. A notable point of



(a) ML-100K dataset



(b) FilmTrust dataset

Fig. 9. Number of steps taken for each episode over 100 episodes for a single user on each dataset.

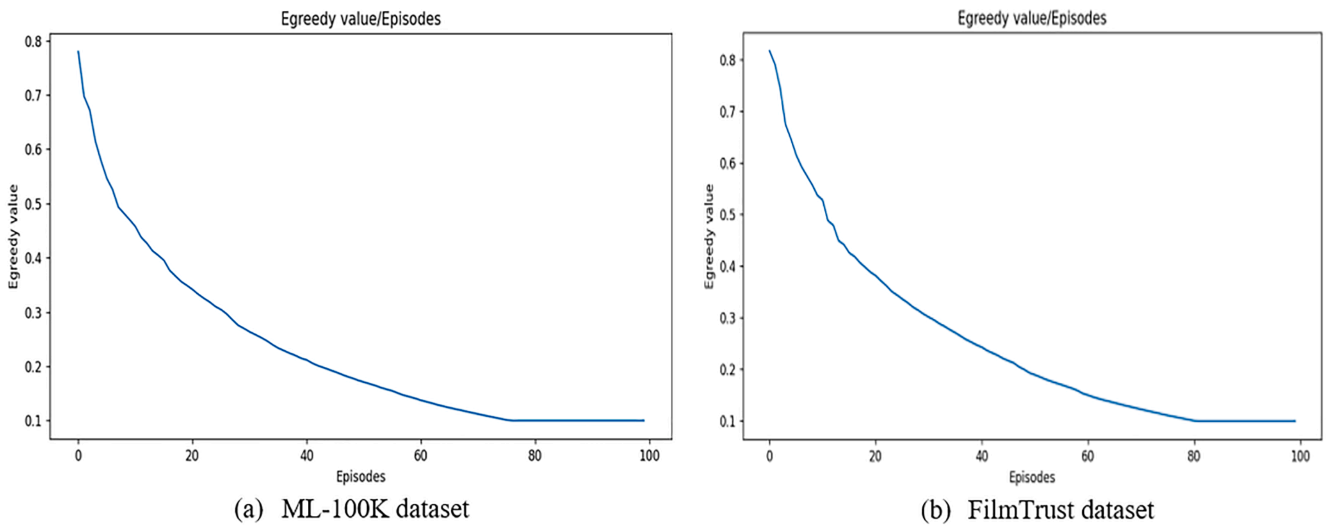


Fig. 10. ∈ -greedy value while agent is learning the environment for both datasets.

inflection is observed at the 75th episode for the ML-100 K dataset and at the 80th episode for the FilmTrust dataset, where the agent's learning curve begins to plateau. Consequently, we can see that the agent's learning is faster with the ML-100 K dataset and slower with the FilmTrust dataset.

5.5. Precision, Recall, Fmeasure and item coverage results

Table 6 presents the results obtained from competitor methods on the ML-100 K dataset. The proposed method and RLRS1 does not present results for the MAE and RMSE evaluation measures as the current methodology does not predict votings, it only predicts items/movies. The precision results of IPWR and ITR are better than all the other methods. The recall and Fmeasure results are best for our proposed method. It can be also observed that for proposed method, precision, recall and F-measure results of SARSA are better than Q-learning implementation, while Q-learning implementation possess higher item coverage than SARSA. Item coverage of our proposed method is not very good in comparison to non-RL methods. This phenomenon can be attributed to the agent's inclination to approach states exhibiting higher degrees of similarity, while concurrently disregarding states characterized by lower similarity ratios.. These low similarity states may contain items that are not present in high similarity states, thus resulting in a reduced item coverage.

Table 7 presents the results for the FilmTrust dataset.. The precision results of IPWR and ITR are better than SPOP and S1. The Precision, Recall and Fmeasure results are best for the proposed method using SARSA implementation. On this dataset, proposed method results using SARSA are better than Q-learning approach. The proposed method's item coverage closely aligns with that of non-RL methods, exhibiting negligible disparities. This may be due to the fact of the agent moving towards states that have higher similarity and ignoring states having lower similarity. These low similar states may contain items that are not present in high similar states thus resulting in a reduced item coverage.

5.6. Impact of random policies on performance

To determine the effectiveness of the learned optimal policy, we defined random policies and measure their performance with the learned optimal policy.

5.6.1. ML-100 K dataset

We performed experiments with the ML-100 K dataset to measure the impact of five random policies having the same policy size for a single random user. These five random policies are given in Table 8 along with the learned policy (policy*). All policies have the same size and all policies in this case have the same start state (21). The action space has the following values: Up = 0, Left = 1, Down = 2 and Right = 3. Policy* denotes the agent's learned policy using the proposed method. It can be observed from Fig. 11, that Policy* has a higher return and Fmeasure value as compared to the other policies. This shows the superiority of our proposed method and the good learning capability of the agent.

We also performed experiments to measure the impact of five random policies having the same policy size for a random user, but with different start states for each policy. The start states are 10, 18, 35, 0 and 15 for each random defined policy, respectively. The results in Fig. 12 show that for the five policies, the return value is lower in comparison to the return value of the proposed method (Policy* in Fig. 11). This indicates that the selection of start states for our proposed algorithm has a good impact on agent earning of rewards/return.

5.6.2. FilmTrust dataset

Various experiments were performed to measure the impact of five random policies having the same policy size for a single random user. These five random policies are shown in Table 9. Moreover, all policies in this case have same start state (17). Action space have following values, Up = 0, Left = 1, Down = 2 and Right = 3. Policy* denotes agent's learned policy using proposed method. It can be observed from

Table 6
Evaluation parameters results of proposed method in conjunction with comparison methods.

	IPWR	ITR	SPOP	S1	RLRS1	Proposed (Q-learning)	Proposed (SARSA)
MAE	0.814	0.825	0.991	1.015	—	—	—
RMSE	1.029	1.039	1.245	1.315	—	—	—
Precision	0.615	0.616	0.477	0.491	0.425	0.481	0.522
Recall	0.349	0.336	0.412	0.405	0.691	0.775	0.758
Fmeasure	0.411	0.402	0.404	0.410	0.518	0.581	0.619
Item Coverage	100	100	92.6	95.2	68.89	76.97	71.70

Table 7
Results on FilmTrust dataset.

	IPWR	ITR	SPOP	S1	RLRS1	Proposed (Q-learning)	Proposed (SARSA)
MAE	0.637	0.625	0.655	0.635	—	—	—
RMSE	0.993	0.83	0.86	0.83	—	—	—
Precision	0.605	0.62	0.55	0.54	0.725	0.759	0.781
Recall	0.545	0.51	0.494	0.515	0.849	0.939	0.958
Fmeasure	0.578	0.57	0.519	0.526	0.765	0.796	0.801
Item Coverage	98.2	99.63	96.1	96.1	89.07	95.07	96.21

Table 8
Five random defined policies.

Policy name	Policy values
Policy*	[3,0,0,1,3,1,3,1,3,1]
Policy1	[0,1,3,2,1,0,1,2,3,1]
Policy2	[3,2,1,2,3,0,1,2,2,2]
Policy3	[0,0,1,1,1,2,2,3,1,0]
Policy4	[1,1,1,1,1,2,2,2,2,2]
Policy5	[0,1,0,1,0,3,2,3,2,3]

Fig. 13, that Policy* have higher return and Fmeasure value as compared to other policies. This shows superiority of the proposed method and good learning capability of agent.

Further experiments are performed to measure the impact of five random policies having same policy size for a random user, but with different start state for each policy. Start state is 10, 15, 20, 25 and 30 for each random defined policy, respectively. Results shown in Fig. 14 shows that for five policies return value is lower as compared to the return value of proposed method (Policy* in Fig. 13). This indicates that selection of start state by the proposed algorithm have good impact on agent earning of rewards/return.

5.7. Impact of the number of episodes on the evaluation measures

In this section, we shall present the impact of the number of episodes on both datasets for different evaluation measures.

5.7.1. ML-100 K dataset

Fig. 15 shows the impact of increased episodes on the evaluation measures. Overall, all evaluations measures show a sinusoidal effect with increased episodes. Fig. 16 indicates that increased episodes have a direct impact on the reward earned by the agent. For each episode, the reward increases except at episodes 70 and 100. After 40 episodes, the average number of steps taken to reach the goal remains constant. Consequently, it can be inferred that the step count per episode remains relatively consistent, and beyond the 40-episode threshold, the agent repeats the same steps, leading to revisits to identical states.

5.7.2. FilmTrust dataset

In the context of the FilmTrust dataset, Fig. 17 illustrates the influence of heightened episodes on the evaluation outcomes. Notably, augmented episodes exhibit negligible effects on Precision and F-

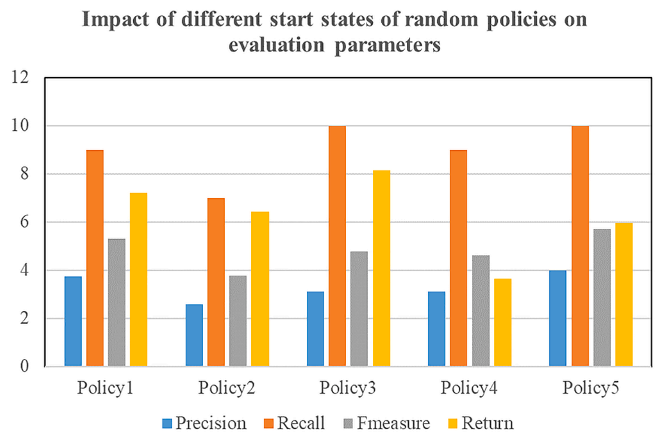


Fig. 12. Evaluation results for five different random policies, in comparison to the learned policy (policy*), with different start states for each policy.

Table 9
Five random defined policies for the FilmTrust dataset.

Policy name	Policy values
Policy*	[0,2,0,2,0,2,0,2,0,2]
Policy1	[0,1,3,2,2,1,1,2,0,1]
Policy2	[1,2,3,2,3,1,1,2,2,2]
Policy3	[1,1,0,0,3,2,2,3,1,0]
Policy4	[1,1,1,1,1,2,2,2,2,2]
Policy5	[0,1,0,1,0,3,2,3,2,3]

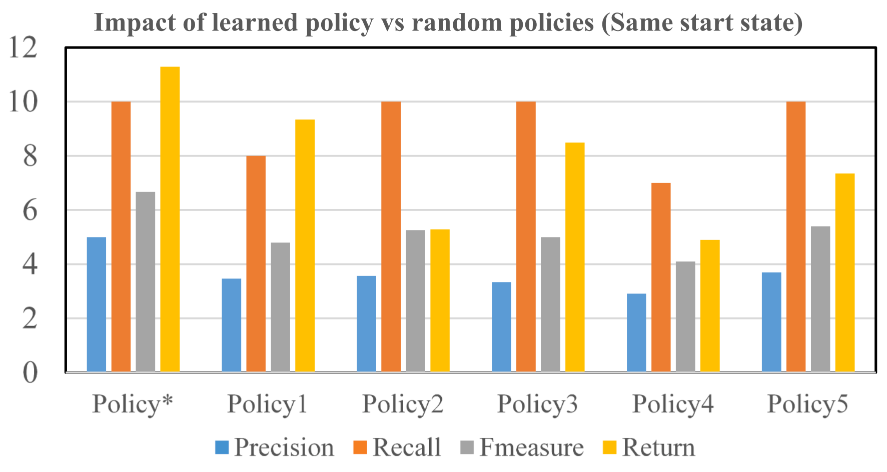


Fig. 11. Evaluation results for five different random policies, in comparison to the learned policy (policy*), with the same start state of (21) for each policy.

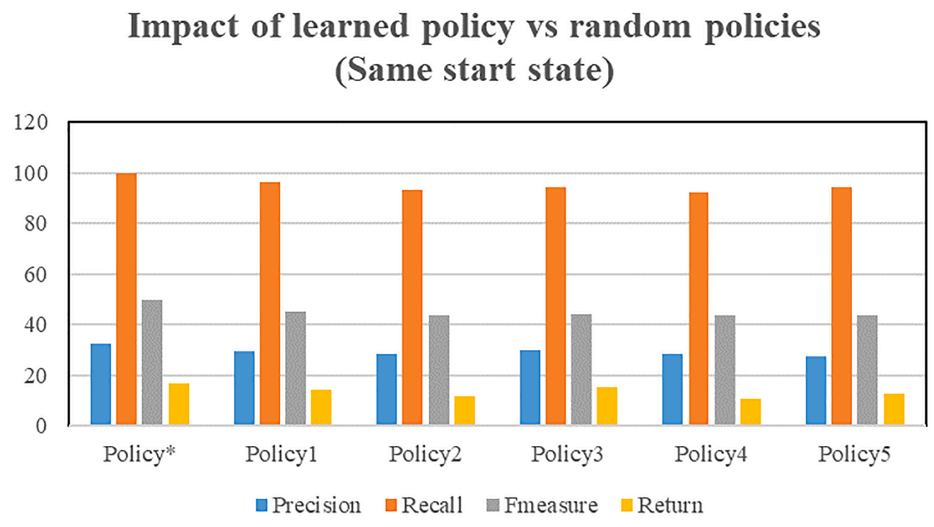


Fig. 13. Evaluation results of learned policy (policy*) **vs** random policies.

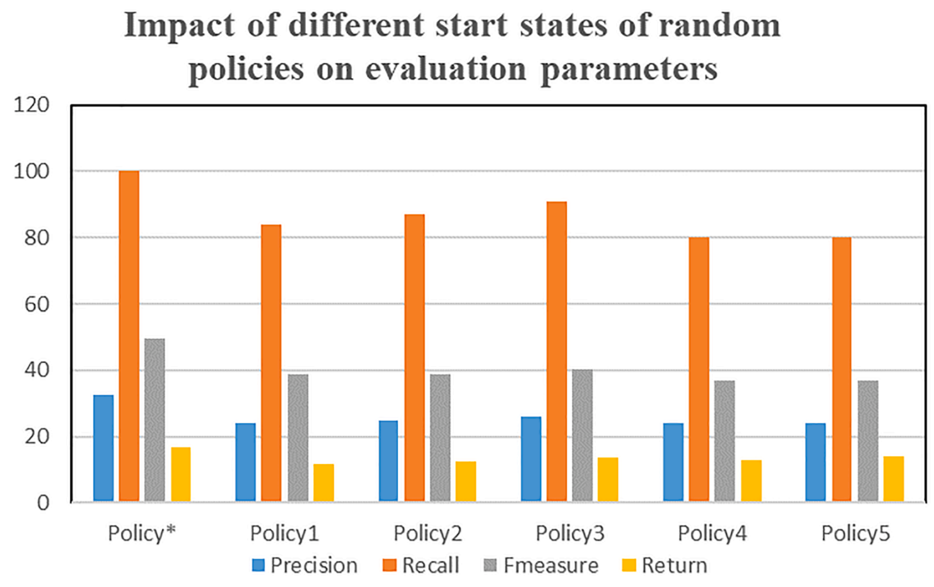


Fig. 14. Evaluation measures results for five different random policies.

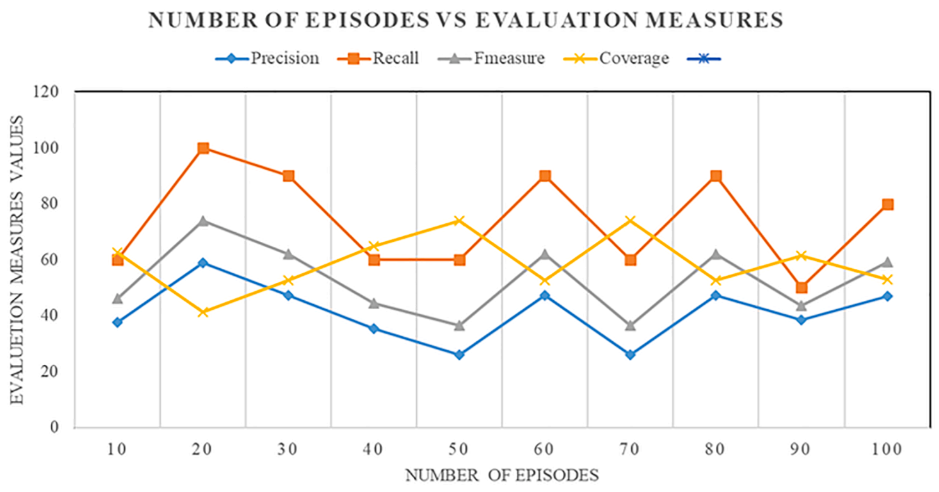


Fig. 15. Impact of increasing number of episodes on the evaluation measures.

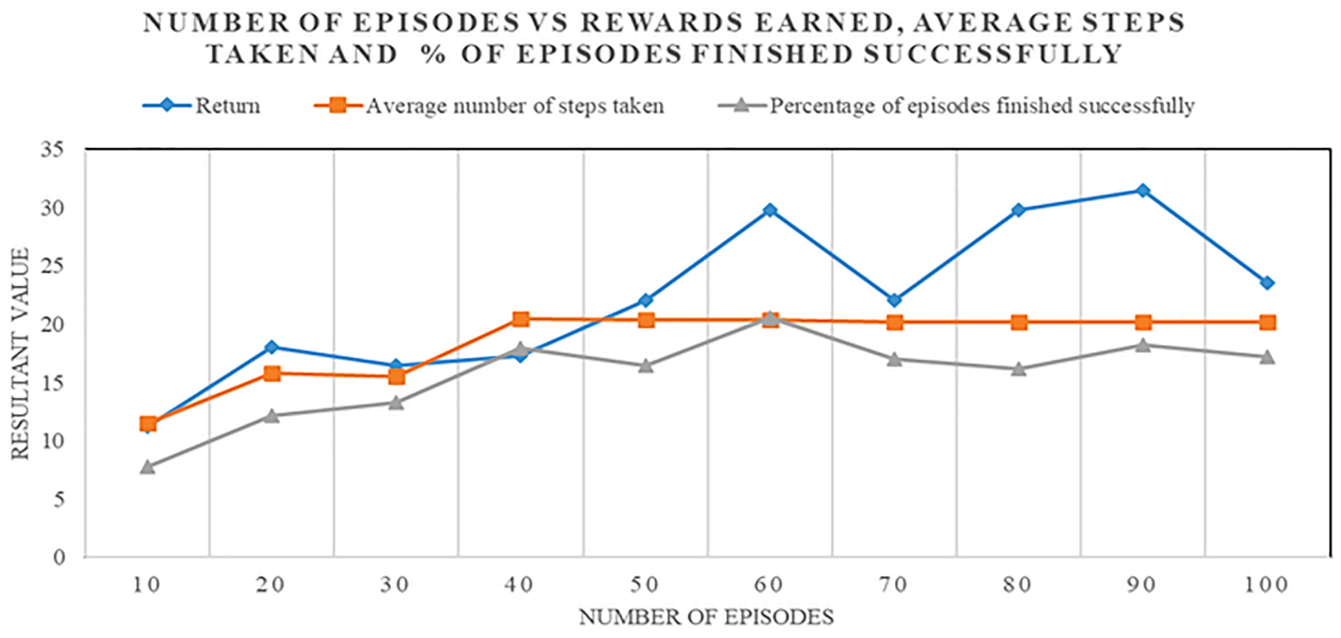


Fig. 16. Impact of increasing number of episodes on Return. The average number of steps taken in each episode and the percentage of episodes finished successfully.

measure, while eliciting some impact on Recall and Coverage. Fig. 18 indicates the impact of increased episodes on the total reward earned, the average number of steps taken in an episode and the percentage of episodes finished successfully. It can be seen that the percentage of episodes finishing successfully in the starting episodes is low and increases as the number of episodes' increases.

In aggregate, around 30 to 35% of episodes conclude successfully at a goal state. The findings unveil that initial episodes exhibit higher return values, indicative of extensive state exploration by the agent. However, this heightened exploration may not necessarily translate to significant improvements in the recommendation process. Delving into the count of steps taken per episode, the initial stages reveal a reduced step count. This phenomenon could stem from the agent's initial lack of familiarity with the environment, potentially leading to encounters with less favorable states. As the agent's understanding of the environment deepens through escalated exploration, the number of steps taken in each episode increases correspondingly.

6. Conclusions and future work

The exponential surge in online data has led to a burgeoning market for Recommender Systems (RSs), accentuating the demand for more precise and contextually sensitive RSs. To address this need, we have reformulated the RS challenge within the framework of Markov Decision Process (MDP), seeking to enhance accuracy and contextual relevance. Biclustering originated from the field of bioinformatics for binding similar rows and columns together. We used biclustering for binding similar rows and columns of user-item votings matrix. These biclusters are then placed on SG to model an RL environment for our recommendation problem. Placement of biclusters on SG is of utmost importance for solving RL problem.

In current work, we used cantor-diagonal traversal method for placement of biclusters on SG. A more accurate placement method can be designed, in which nearby biclusters are more similar to each other, while biclusters at farther grid position should be more dissimilar. For this purpose, advance grid positioning methods like Hilbert curve or Z-

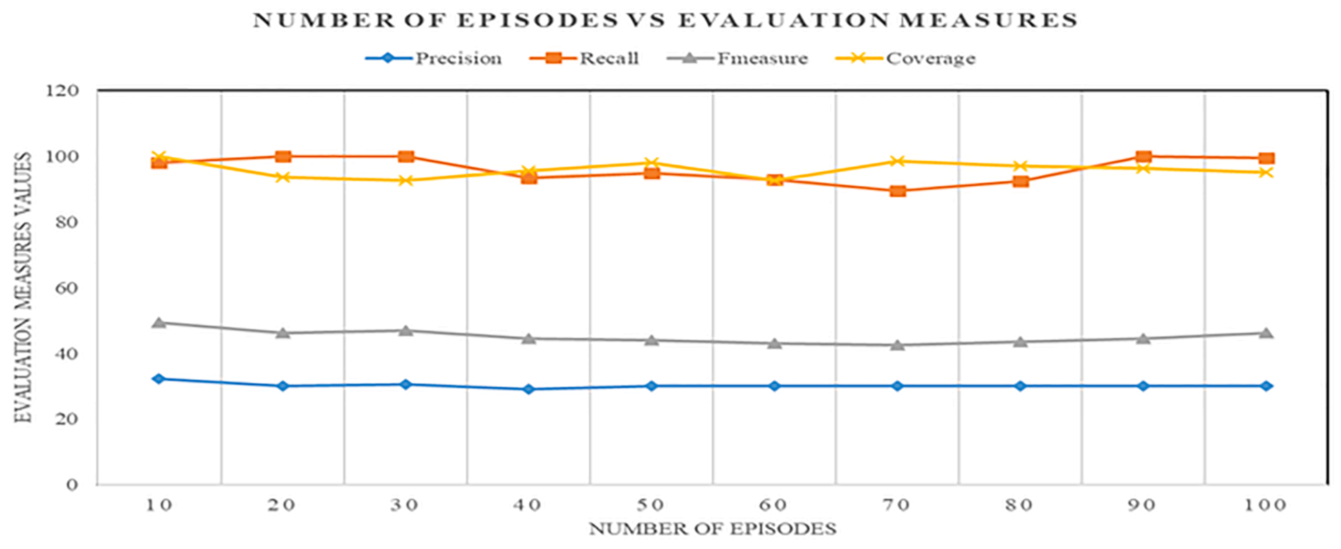


Fig. 17. Impact of increasing number of episodes on evaluation measures.

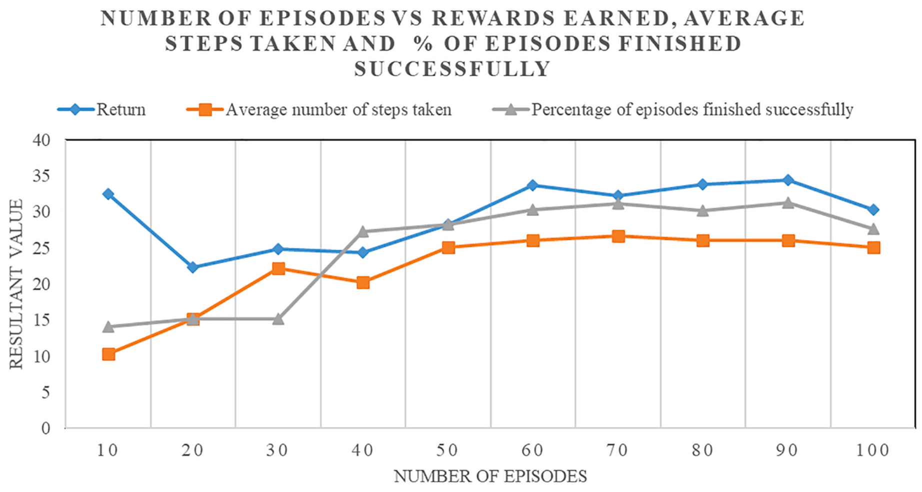


Fig. 18. Impact of number of episodes on Return, steps and successful episodes.

order traversal are suggested. In this work, we used Jaccard measure to compute rewards. Reward value can be same for different states, thus making agent indecisive about where to move. A more accurate reward measuring function that may consider demographic information of users and movies can also be designed. In this work, we used SMSR as a fitness function to measure quality of biclusters. SMSR is not able to identify a perfect correlation among rows or columns of a bi-cluster. A correlation based fitness function such as PCC can be more effective as it does not emphasize on specific magnitudes like SMSR and considers both positive and negative correlations. Action space is limited to only four actions; action space can be increased to include diagonal movement. In current work, to make user-item votings matrix binary, missing votings are replaced with a value of zero, while available voting values are replaced with one. Thus making low and high voting values indiscriminative. A tri-clustering algorithm can be modelled to handle this situation. This may change overall scenario like position of tri-clusters on the squared grid, a new fitness function that should be able to measure quality of tri-clusters. We evaluated our proposed method on MovieLens and Film-Trust datasets and it is observed that our proposed method outperformed competitor methods that include RL methods, pure CF methods and clustering methods. Also our algorithm achieved a better start state that yields an optimal policy to achieve the goal.

As a potential avenue for future research, contextual reinforcement learning could be extended within the existing RL framework to encompass contextual information encompassing user demographics, social networking information, temporal context, and item characteristics. This extension has the potential to yield recommendations that are both highly personalized and contextually relevant, thereby enriching the user experience.

7. Funding information

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

CRediT authorship contribution statement

Arta Iftikhar: Conceptualization. **Mustansar Ali Ghazanfar:** Supervision, Data curation. **Mubbashir Ayub:** Conceptualization, Methodology, Software, Visualization, Investigation, Writing – review & editing. **Saad Ali Alahmari:** Writing – review & editing. **Nadeem Qazi:** Software, Validation. **Julie Wall:** Writing – review & editing.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

Afsar, M. M. (2022). *Personalized recommendation using reinforcement learning (Unpublished doctoral thesis)*. Calgary, AB: University of Calgary.
Ahn, H. J. (2008). A new similarity measure for collaborative filtering to alleviate the new user cold-starting problem. *Information Sciences, 178*(1), 37–51.
Ayub, M., Ghazanfar, M. A., Mehmood, Z., Alyoubi, K. H., & Alfakeeh, A. S. (2019). Unifying user similarity and social trust to generate powerful recommendations for smart cities using collaborating filtering-based recommender systems. *Soft Computing*, 1–24.
Ayub, M., Ghazanfar, M. A., Mehmood, Z., Saba, T., Alharbey, R., Munshi, A. M., & Alrige, M. A. J. P.o. (2019). Modeling user rating preference behavior to improve the performance of the collaborative filtering based recommender systems. *PLoS One, 14* (8), e0220129.
Bobadilla, J., Ortega, F., Hernando, A., & Gutiérrez, A. (2013). Recommender systems survey. *Knowledge-based Systems, 46*, 109–132.
Bohnenberger, T., & Jameson, A. (2001). *When policies are better than plans: Decision-theoretic planning of recommendation sequences.* Paper presented at the Proceedings of the 6th international conference on Intelligent user interfaces.
Candan, K. S., & Sapino, M. L. (2010). *Data management for multimedia retrieval.* Cambridge University Press.
Cheng, Y., & Church, G. M. (2000). Biclustering of expression data. In intelligent systems for molecular biology. In: Menlo Park: AAAI Press.
Choi, S., Ha, H., Hwang, U., Kim, C., Ha, J.-W., & Yoon, S. (2018). Reinforcement learning based recommender system using biclustering technique. *arXiv preprint arXiv:1801.05532*.
Chowdhury, G. G. (2010). *Introduction to modern information retrieval*: Facet publishing.
Costa, A., & Roda, F. (2011). *Recommender systems by means of information retrieval.* In: Paper presented at the Proceedings of the International Conference on Web Intelligence, Mining and Semantics.
Ejaz, W., & Anpalagan, A. (2019a). Communication technologies and protocols for internet of things. In *Internet of things for smart cities* (pp. 17–30). Springer.
Ejaz, W., & Anpalagan, A. (2019b). Internet of things for smart cities: Overview and key challenges. *Internet of Things for Smart Cities*, 1–15.
Ekstrand, M. D., Riedl, J. T., & Konstan, J. A. (2011). *Collaborative filtering recommender systems.* Now Publishers Inc.
Fkih, F. (2022). Similarity measures for Collaborative Filtering-based Recommender Systems: Review and experimental comparison. *Journal of King Saud University-Computer and Information Sciences, 34*(9), 7645–7669.
Gao, C., Xu, K., Zhou, K., Li, L., Wang, X., Yuan, B., & Zhao, P. (2022). *Value penalized Q-learning for recommender systems.* Paper presented at the Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval.
Ge, Y., Zhao, X., Yu, L., Paul, S., Hu, D., Hsieh, C.-C., & Zhang, Y. (2022). *Toward Pareto efficient fairness-utility trade-off in recommendation through reinforcement learning.* Paper presented at the Proceedings of the fifteenth ACM international conference on web search and data mining.
Hammad, R., & Ludlow, D. (2016). *Towards a smart learning environment for smart city governance.* Paper presented at the Proceedings of the 9th international conference on utility and cloud computing.

Hartigan, J. A. (1972). Direct clustering of a data matrix. *Journal of the American Statistical Association, 67*(337), 123–129.

Hu, B., Shi, C., & Liu, J. (2017). *Playlist recommendation based on reinforcement learning.* Paper presented at the International Conference on Intelligence Science.

Iftikhar, A., Ghazanfar, M. A., Ayub, M., Mehmood, Z., & Maqsood, M. (2020). An improved product recommendation method for collaborative filtering. *IEEE Access, 8*, 123841–123857.

Intayoad, W., Kamyod, C., & Temdee, P. (2018). *Reinforcement learning for online learning recommendation system.* Paper presented at the 2018 Global Wireless Summit (GWS).

Jannach, D., Zanker, M., Felfernig, A., & Friedrich, G. (2010). *Recommender systems: An introduction.* Cambridge University Press.

Joachims, T., Freitag, D., & Mitchell, T. (1997). *Webwatcher: A tour guide for the world wide web.* Paper presented at the IJCAI (1).

Khalid, A., Ghazanfar, M. A., Azam, M. A., Aldhafiri, Y. F., & Zahra, S. (2017). Scalable and practical One-Pass clustering algorithm for recommender system. *Intelligent Data Analysis, 21*(2), 279–310.

Konstan, J. A., Miller, B. N., Maltz, D., Herlocker, J. L., Gordon, L. R., & Riedl, J. (1997). GroupLens: Applying collaborative filtering to Usenet news. *Communications of the ACM, 40*(3), 77–87.

Lee, S. (2017). *Improving jaccard index for measuring similarity in collaborative filtering.* Paper presented at the International Conference on Information Science and Applications.

Li, W., Wang, G.-G., & Gandomi, A. H. J. A. o. C. M. i. E. (2021). A survey of learning-based intelligent optimization algorithms. *28*, 3781–3799.

Li, G., Wang, G.-G., Dong, J., Yeh, W.-C., & Li, K. J. I. s. (2021). DLEA: A dynamic learning evolution algorithm for many-objective optimization. *574*, 567–589.

Liebman, E., Saar-Tsechansky, M., & Stone, P. (2014). Dj-mc: A reinforcement-learning agent for music playlist recommendation. *arXiv preprint arXiv:1401.1880*.

Liu, C., & Malik, H. (2014). *A new investment strategy based on data mining and neural networks.* Paper presented at the 2014 International Joint Conference on Neural Networks (IJCNN).

Liu, H., Hu, Z., Mian, A., Tian, H., & Zhu, X. (2014). A new user similarity model to improve the accuracy of collaborative filtering. *Knowledge-based Systems, 56*, 156–166.

Lu, Z., & Yang, Q. (2016). Partially observable markov decision process for recommender systems. *arXiv preprint arXiv:1608.07793*.

Mahmood, T., & Ricci, F. (2007). *Learning and adaptivity in interactive recommender systems.* Paper presented at the Proceedings of the ninth international conference on Electronic commerce.

Mahmood, T., & Ricci, F. (2009). *Improving recommender systems with adaptive conversational strategies.* Paper presented at the Proceedings of the 20th ACM conference on Hypertext and hypermedia.

Mahmood, T., Mujtaba, G., & Venturini, A. (2014). Dynamic personalization in conversational recommender systems. *Information Systems and e-Business Management, 12*(2), 213–238.

Malik, H. (2013). Acoustic environment identification and its applications to audio forensics. *IEEE Transactions on Information Forensics and Security, 8*(11), 1827–1837.

Mobasher, B., Cooley, R., & Srivastava, J. (2000). Automatic personalization based on web usage mining. *Communications of the ACM, 43*(8), 142–151.

Mukhopadhyay, A., Maulik, U., & Bandyopadhyay, S. (2009). A novel coherence measure for discovering scaling biclusters from gene expression data. *Journal of Bioinformatics and Computational Biology, 7*(05), 853–868.

Prakash, H. R., Korostenskaja, M., Lee, K., Baumgartner, J., Castillo, E., & Bagci, U. (2017). *Automatic response assessment in regions of language cortex in epilepsy patients using ECoG-based functional mapping and machine learning.* Paper presented at the 2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC).

Prelić, A., Bleuler, S., Zimmermann, P., Wille, A., Bühlmann, P., Gruissem, W., … Zitzler, E. (2006). A systematic comparison and evaluation of biclustering methods for gene expression data. *Bioinformatics, 22*(9), 1122–1129.

RaviPrakash, H., Korostenskaja, M., Castillo, E. M., Lee, K. H., Salinas, C. M., Baumgartner, J., … Bagci, U. (2020). Deep Learning provides exceptional accuracy to ECoG-based Functional Language Mapping for epilepsy surgery. *Frontiers in Neuroscience, 14*, 409.

Rodriguez-Baena, D. S., Perez-Pulido, A. J., & Aguilar– Ruiz, J. S. (2011). A biclustering algorithm for extracting bit-patterns from binary datasets. *Bioinformatics, 27*(19), 2738–2745.

Rojanavasu, P., Srinil, P., & Pinngern, O. (2005). New recommendation system using reinforcement learning. *Special Issue of the Intl. J. Computer, the Internet and Management, 13*(SP 3).

Sargar, R. B. (2020). *Recommender system using reinforcement learning.* Arizona State University.

Sarwar, B. M., Karypis, G., Konstan, J., & Riedl, J. (2002). *Recommender systems for large-scale e-commerce: Scalable neighborhood formation using clustering.* Paper presented at the Proceedings of the fifth international conference on computer and information technology.

Shani, G., Heckerman, D., Brafman, R. I., & Boutilier, C. (2005). An MDP-based recommender system. *Journal of Machine Learning Research, 6*(9).

Silveira, T., Zhang, M., Lin, X., Liu, Y., Ma, S. J. I. J. o. M. L., & Cybernetics. (2019). How good your recommender system is? A survey on evaluations in recommendation. *10*, 813–831.

Srivihok, A., & Sukonmanee, P. (2005). *E-commerce intelligent agent: personalization travel support agent using Q Learning.* Paper presented at the Proceedings of the 7th international conference on Electronic commerce.

Stamenkovic, D., Karatzoglou, A., Arapakis, I., Xin, X., & Katevas, K. (2022). *Choosing the best of both worlds: Diverse and novel recommendations through multi-objective reinforcement learning.* Paper presented at the Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining.

Sutton, R. S., & Barto, A. G. (1998). Introduction to reinforcement learning: Book.

Sutton, R. S., & Barto, A. G. (1998a). *Introduction to reinforcement learning, 135*. MIT press Cambridge.

Taghipour, N., Kardan, A., & Ghidary, S. S. (2007). *Usage-based web recommendations: a reinforcement learning approach.* Paper presented at the Proceedings of the 2007 ACM conference on Recommender systems.

Vodopivec, T., Samothrakis, S., & Ster, B. (2017). On monte carlo tree search and reinforcement learning. *Journal of Artificial Intelligence Research, 60*, 881–936.

Watkins, C. J. C. H. (1989). Learning from delayed rewards. Doctoral Thesis, King's College.

Xin, X., Karatzoglou, A., Arapakis, I., & Jose, J. M. (2020). *Self-supervised reinforcement learning for recommender systems.* Paper presented at the Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval.

Yip, K. Y., Cheung, D. W., & Ng, M. K. (2004). Harp: A practical projected clustering algorithm. *IEEE Transactions on Knowledge and Data Engineering, 16*(11), 1387–1397.

Zahra, S., Ghazanfar, M. A., Khalid, A., Azam, M. A., Naeem, U., & Prugel-Bennett, A. (2015). Novel centroid selection approaches for KMeans-clustering based recommender systems. *Information sciences, 320*, 156–189.

Zou, L., Xia, L., Ding, Z., Yin, D., Song, J., & Liu, W. (2019). *Reinforcement learning to diversify top-n recommendation.* Paper presented at the International Conference on Database Systems for Advanced Applications.