KI - Künstliche Intelligenz
https://doi.org/10.1007/s13218-019-00620-2

TECHNICAL CONTRIBUTION

Matrix- and Tensor Factorization for Game Content Recommendation

Rafet Sifa[1] · **Raheel Yawar**[2] · **Rajkumar Ramamurthy**[1] · **Christian Bauckhage**[1] · **Kristian Kersting**[3]

Received: 6 October 2018 / Accepted: 6 September 2019

Abstract
Commercial success of modern freemium games hinges on player satisfaction and retention. This calls for the customization of game content or game mechanics in order to keep players engaged. However, whereas game content is already frequently generated using procedural content generation, methods that can reliably assess what kind of content suits a player's skills or preferences are still few and far between. Addressing this challenge, we propose novel recommender systems based on latent factor models that allow for recommending quests in a single player role-playing game. In particular, we introduce a tensor factorization algorithm to decompose collections of bipartite matrices which represent how players' interests and behaviors change over time. Extensive online bucket type tests during the ongoing operation of a commercial game reveal that our system is able to recommend more engaging quests and to retain more players than previous handcrafted or collaborative filtering approaches.

Keywords Player retention · Recommender systems · Latent factor models

1 Introduction

Content recommender systems are by now established tools in e-commerce where they provide personalized suggestions to users who are browsing vast portfolios of products [25]. Players of digital games, too, face numerous options, both in-game (e.g. choosing quests, characters, or tactics) and out-game (e.g. buying downloadable content in semi persistent games, making in-app purchases in free-to-play (F2P) games, or switching to other games on online gaming platforms) [14, 16, 17, 20]. Their decisions impact gameplay experience which, accumulated over time, results in either player retention or churn. Game developers are therefore interested in building recommendation systems to provide players with personalized game content so as to keep them engaged [20].

In this paper we deal with recommender systems for game content and focus on the problem of recommending quests in an F2P game called *Trollhunters: Adventures in the Troll Caves* (see Fig. 1). In addition to the scripted main quests which progress this game, players are offered side quests which are generated via procedural content generation. To keep players engaged and to increase retention, the difficulty of these side quests should match players' skills so as to maintain a feeling of flow [2]. In other words, the difficulty should neither be so high that players give up nor should it be so low that players get bored from lack of challenge.

Other factors, too, contribute to the level of enjoyment. These include quest type, quest duration, weapons available in a quest, etc. However, identifying the interplay of these factors and their combined impact on player satisfaction is a non-trivial problem. Moreover, given that the game has a large active player base, any meaningful analysis cannot be done manually but requires automatic analytics tools. We therefore consider data-driven approaches such as learning representations from player-quest interactions and build a recommender system based on latent factor models.

Developing an in-game quest recommender system for *Trollhunters: Adventures in the Troll Caves* poses a number of challenges. Primarily, due to the large number of active players of this Web-based game, any useful recommender system has to be trained incrementally in an online manner. Moreover, the

✉ Christian Bauckhage
christian.bauckhage@iais.fraunhofer.de

Rafet Sifa
rafet.sifa@iais.fraunhofer.de

[1] Fraunhofer IAIS, Sankt Augustin, Germany

[2] Flying Sheep Studios, Cologne, Germany

[3] Department of Computer Science, TU Darmstadt, Darmstadt, Germany

Published online: 13 September 2019

(a) interaction with a quest giver **(b)** the quest dialogue **(c)** a combat scene

Fig. 1 Screenshots from the game *Trollhunters: Adventures in the Troll Caves* (in German: *Trolljäger: Abenteuer in den Trollhöhlen*). Its core loop consists in choosing a quest offered by various NPCs and entering the dungeon to complete all phases of a chosen quest. The individual panels show: **a** one of the three NPC quest givers (the exclamation point and circle indicate that a quest is available); **b** the quest dialog box for the quest phases (three in this case) and steps of each phase; **c** the protagonist and an AI companion (in the back) combating two trolls (on the right)

game was developed along with facilities for content recommendation in mind and rudimentary solutions were bundled with the game engine and released to the users as a one-time deliverable. This means that the traditional process of building a recommender system using the *build, test, and tune* approach is not a viable option in our case; rather, any more advanced solution has to comply with existing APIs. Finally, baseline evaluation metrics used in off-line batch testing settings are not well suited for online games.

Addressing these challenges, we describe novel tensor factorization methods for content recommendation that can be be trained iteratively. In order to evaluate our models during deployment of the game, we perform bucket testing on nine different user groups of 2854 unique players each (our overall test population thus consist of 25,686 players) and consider player retention as our main evaluation metric as it implicitly measures player engagement and directly impacts monetization. We also evaluate three more game specific metrics to further assess the quality of the recommendations produced by our approach. Our tests consistently reveal that quest recommendation based on tensor factorization methods outperforms all other methods and, in particular, yields better results than an intricate, handcrafted approach.

Next, we briefly review the related literature and contrast it to the work presented here. We then discuss details as to the game that forms the basis for our study and discuss the handcrafted quest recommender originally used in the game. After that, we recall the ideas behind collaborative filtering systems and discuss matrix factorization and our novel, tensor factorization based approach to quest recommendation before we present the results of our online evaluation.

2 Related Work and Own Contributions

The idea of using AI to manage the pacing of a game is well established [18, 31]. Several commercial games use so called AI directors that apply models of players' stress levels and adjust difficulty correspondingly. Here, however, we are mainly concerned with explicit content recommendation rather than with implicit content adaptation.

Prior work on game content recommendation can roughly be categorized according to the kind of data used to generate recommendations. In particular, most previous approaches have either considered contextual meta data or in-game behavioral data.

Contextual data for game recommendation is often textual and mainly comprises game reviews or comments found in game related online forums. Approaches considering such data typically build on vector space embeddings of text. For instance, the work in Ref. [13] represents user reviews in a vector space that reflects frequencies of co-occurrences of adjectives and gaming related terms. The authors then apply information theoretic approaches to co-cluster reviews and games in order to generate recommendations and conduct an offline evaluate their approach on a group of ten players. The authors of Ref. [16] crawl game related terms from the Web and construct a "GameNet" which allows for searching for similar games using techniques akin to those employed by search engines. Extending this architecture, the authors utilize matrix factorization to build a context-based recommender system that recommends games based on reviews [15]. Surveying a

group of ten players, they evaluate their results and conclude that the latent factor model outperforms their baseline recommender.

While user reviews easily allow for identifying similar games of similar quality for recommendation, they are less well suited for the identification of preferred game mechanics or individual playing styles. Another line of work is thus concerned with game content recommendation based on in-game behavioral data [4, 19].

The authors of Ref. [20] use constrained matrix factorization model to build a game recommender system based on playtime information which, unlike explicit game ratings or reviews, is a form of *implicit feedback* that players provide subconsciously. The authors evaluate the generalization capabilities of their methods in an off-line fashion by predicting playing times on a set of holdout games. Dealing with multiplayer online games, a recent industrial case study reported in Ref. [29] applies neighborhood based recommendation which ranks a player's profile with respect to those of other players. Extending such baseline approaches, the authors of Ref. [23] consider behavioral in-game data and apply matrix factorization based representation learning to build recommender systems for three different behavioral aspects; they evaluate their using a questionnaire survey where they ask 30 about their motivations and preferences.

The work reported here differs from previous approaches to game content recommendation both in methodology and scale. To the best of our knowledge, our work is the first to integrate tensor factorization based collaborative filtering models into a commercial online game with a player base of hundreds of thousands of registered players. We are further not aware of prior work that provide an online evaluation by bucket testing the performance of (the settings of) different recommendation algorithms using groups of several thousand players each.

3 Trollhunters: Adventures in the Troll Caves

We developed and evaluated our game content recommender algorithms for the game *Trollhunters: Adventures in the Troll Caves*[1]. This is an ad-driven F2P role-playing dungeon crawler (see Fig. 1) where players control a protagonist who is accompanied by two non-player characters (NPCs) controlled by an AI component.

The core loop of the game consists of the player starting in a marketplace, choosing a quest offered by an NPC, entering a procedurally generated dungeon, completing the quest, and returning to the marketplace. Quest phases may include finding gems, fighting enemies, and, occasionally, finding lost friends. Upon completion of a quest, players gain experience points and level up. Points can be invested into three player attributes, namely strength (health), agility (movement and attack speed), and damage (dealt to opponents). The player character maxes out at level 30 and can invest only ten points into each attribute. Quest consist of one to three phases and are classified into two types: story quests and side-quests where the latter are generated on the fly using procedural content generation. This, in turn, depends on a difficulty value and on random seed parameters where the difficulty value impacts enemy strength. The quest generator uses pre-built dungeon pieces and in-game items and assembles them into levels; in total, it allows for the creation of 85,000 different quests.

New players are first guided through a tutorial quest in order to familiarize them with the game and to build an initial player profile. Up until they reach level 3, players are given random quests; players of higher skill levels receive personalized quest suggestions generated by a recommender engine. Players complete a quest if they finish all its phases but they may also fail it if they die. Crucially, however, players can also abandon or quit a quest via the in-game menu if they dislike it and a development goal is to minimize abandonment rates. Using a client-server architecture, the game is rendered at the client side and recommendation algorithms run at the server side.

4 Content-Based Quest Recommendation

The content-based (CB) quest recommendation solution originally employed in the game is an intricate, handcrafted, iteratively designed rule-based approach inspired by Ref. [12] that consists of a profile learner and a filtering component. The primary task of the profile learner is to find the most suitable quest for players based on their skills. Its secondary task is to use players' preferences to determine likely most preferred quests. We work on the assumption that the true skill of a player is not always deducible from the player's progress in the game and we therefore attempt to predict which level of difficulty would be suitable for individual players based on their performance in the quests they have previously played.

To estimate a player's skill, we apply linear smoothing to the attributes of "Accuracy", "Health Lost", and "Failure Rate" recorded in all quests completed so far. In particular, we assign higher weights to the most recent recordings because, as players learn the game mechanics, their performance will improve over time. On the other hand, a player returning to the game after a hiatus might perform less well than previously, so we would like to reduce the difficulty before ramping it up again.

[1] The game is an HTML5 application hosted on Toggo: http://www.toggo.de/serien/trolljaeger/index-4310.htm

Given a total of m_p quests completed by player p and weights $w_j \in [0, 1]$ for the j-th quest, we compute weighted averages a_p and h_p for "Accuracy" and "Health Lost" for each player. We then compute the mean μ_a and standard deviation σ_a of the accuracy values of all players. However, for the mean health loss value of the population, we manually set $\mu_h = 40\%$ and compute the standard deviation σ_h w.r.t. this choice. This heuristic resulted from several rounds of playtesting and reflects the idea that a difficulty level should neither be too easy nor too hard.

Weighted average failure rates r_p for player p are computed in a slightly different manner. We consider the ratio of quests failed f_l and quests tried t_l at level l and let

$$r_p = \sum_{l=1}^{L} w_l \frac{f_l}{t_l} \tag{1}$$

where $L = 30$ is the maximum level a player can reach and $w_l = l \Big/ \sum_{i=1}^{L} i$ is the normalization coefficient. Again, we compute the mean μ_r and standard deviation σ_r of the accuracy values of all players.

Given these statistics, we compute a personalized difficulty value for each player by measuring how much better or worse the player performs compared to the average. This difference allows us to compute how much the player's preferred difficulty value might vary from the player's level. In case of "Accuracy" and "Health Lost", if the player's values are above the global value, the difference is positive; if the player's values are below, the difference is negative. However, the "Failure Rate" only influences the difficulty difference if it drops below the global value; otherwise, the difference is set to zero.

Given the three differences, we take their weighted sum where the weights are set to 18.33%, 15%, and 66.67% for "Accuracy", "Health Lost", and "Failure Rate", respectively. These were determined from extensive play-testing experiments. The weighted (positive or negative) sum is finally added to the player's level to determine a difficulty value. This value, in turn, is stored as part of the player information and is recomputed each time the player finishes another quest. Whenever the player needs a fresh batch of quests to choose from, we recommend quests according to player's difficulty value rather than according to the player's level.

The secondary function of our content-based approach is to predict preferred quest types. A quest can take place in one out of two environments and can have up to three unique phases. To create the user's preference profile, we divide the number of types of quests played by the total number of quest played (considering the finished quests). This provides personalized weights for the different types of quests and allows for a weighted selection of quests presented to the player. A potential pitfall of this method and any other content-based approach, in general, is over-specialization [12]. To avoid this, 30% of the quests presented to the player are chosen randomly.

5 Collaborative Filtering for Quest Recommendations

A fundamental challenge for rule-based content recommendation in games is due to the wide variety of possible player behaviors. The multitude of individual playing styles and preferences renders it neigh impossible to maintain comprehensive, handcrafted sets of recommendation rules for most modern games; even slight changes in the game mechanics (due to updates) or the player demographics usually require intricate rule adjustments and numerous iterations of playtesting to re-balance suggestions and to minimize player churn. Data-driven methods such as collaborative filtering [8, 11] attempt to circumvent these issues by means of automatically mining in-game data and training models of types of player behavior and preferences [6, 23–25, 30].

In the context of games, neighborhood oriented collaborative methods have been widely adopted due to their ease of implementation and practical success [15, 20, 29]. The main idea behind such methods is to recommend content to players based on matching their in-game behavioral data against those of other players. Similar players are typically found by considering similarity measures in a vector space defined over a number of content features determined by the game developers. Examples of recommender systems that make use of representations (of players) in vector spaces include the afore mentioned approaches to game review based or playtime based approaches [13, 15, 20].

Considering the task of quest recommendation in our game, we note that we do not have access to any explicit feedback that would allow for quantifying a player's satisfaction with the quests played. Hence, we consider implicit feedback in terms of behavioral data recorded while players are questing.

Unlike conventional single item-user representations, we consider a more general approach and adopt a third-order tensor representation [9, 24, 32] by grouping different bipartite matrices containing relations certain behavioral features. That is, given n players, m quests and d different behavioral features that are related to the played quests, we consider a collection of matrices

$$\mathcal{X} = \{\boldsymbol{X}_1, \ldots, \boldsymbol{X}_d\} \tag{2}$$

where each slice $\boldsymbol{X}_c \in \mathbb{R}^{m \times n}$ and x_{cji} represents the performance of the c-th feature in the j-th quest played by the i-th player. Here, we consider $d = 7$ features, namely "User Level", "Strength", "Agility", "Damage", "Accuracy", "Health Lost", and "Completion Time".

5.1 Neighborhood Oriented Recommendations

In a simple approach, we can then build a neighborhood oriented (NO) recommender system by calculating the cosine similarity $s(\cdot,\cdot)$ between two arbitrary players for each content matrix $\boldsymbol{X}_c$ and combine the results as a weighted sum over slices by considering the composite similarity

$$\hat{s}(\boldsymbol{X}_{::i}, \boldsymbol{X}_{::j}) = \sum_{c=1}^{d} w_c \; s(\boldsymbol{x}_{c:i}, \boldsymbol{x}_{c:j}), \tag{3}$$

where $\boldsymbol{w} \in \mathbb{R}^d$ is a weight vector, which contains coefficients for a convex combination of the similarity values for each slice (i.e $\boldsymbol{w} \geq \boldsymbol{0}$ and $\boldsymbol{w}^T\boldsymbol{1} = 1$), $\boldsymbol{X}_{::i} \in \mathbb{R}^{m \times d}$, represents the matrix containing the i-th column of each slice, and $\boldsymbol{x}_{c:i} \in \mathbb{R}^m$ represents the i-th column of the c-th slice of $\mathcal{X}$.

5.2 Matrix Factorization Based Recommender Systems

A known characteristic of neighborhood oriented methods is that they tend to assign high similarity values to players that have only played the same games together and might not produce recommendations for similar quests. To circumvent this issue, we may use matrix factorization methods that can reveal the hidden structures in the dataset of quests in that they assign similar weights to similar quests in a *latent factor* space. Traditionally, this is often realized by *individually* factorizing each slice of $\boldsymbol{X}_i$ into two factor matrices [3, 5, 20] by considering the truncated singular value decomposition

$$\boldsymbol{X}_i = \boldsymbol{U}_i \, \boldsymbol{\Sigma}_i \, \boldsymbol{V}_i^T = \boldsymbol{U}_i \, \boldsymbol{\Sigma}_i^{\frac{1}{2}} \boldsymbol{\Sigma}_i^{\frac{1}{2}} \, \boldsymbol{V}_i^T = \boldsymbol{Q}_i \boldsymbol{P}_i, \tag{4}$$

where, for $k < rank(\boldsymbol{X}_i)$, the diagonal $\boldsymbol{\Sigma}_i \in \mathbb{R}^{k \times k}$ contains the value-sorted highest k singular values and $\boldsymbol{U}_i \in \mathbb{R}^{m \times k}$ and $\boldsymbol{V}_i \in \mathbb{R}^{n \times k}$ are respectively the truncated left and right basis matrices (containing the *first* k columns of each basis matrix). With respect to the rightmost expression in (4), we note that $\boldsymbol{Q}_i \in \mathbb{R}^{m \times k}$ is the factor matrix for quests and $\boldsymbol{P}_i \in \mathbb{R}^{k \times n}$ is the factor matrix for players.

Similar to neighborhood oriented recommendation, we may then stack the individual matrices $\boldsymbol{P}_i$ into a tensor

$$\mathcal{P} = \{\boldsymbol{P}_1, \ldots, \boldsymbol{P}_d\} \tag{5}$$

and determine similarities between players by means of evaluating (3) on $\mathcal{P}$. In the remainder of this paper, we will refer to this recommendation method as the matrix factorization (MF) approach.

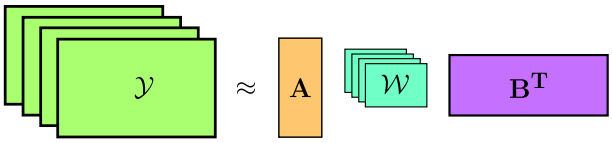


Fig. 2 RTDD of a collection of bipartite matrices for game content recommendations. The model factors each slice $\boldsymbol{Y}_i \in \mathbb{R}^{m \times n}$ of the data tensor $\mathcal{Y}$ as a product of two global basis matrices $\boldsymbol{A} \in \mathbb{R}^{m \times p}$ and $\boldsymbol{B} \in \mathbb{R}^{n \times q}$ and a corresponding local weighting matrix $\boldsymbol{W}_i \in \mathbb{R}^{p \times q}$

6 Tensor Factorization for Quest Recommendation

While the above matrix factorization approach can capture latent similarities between available quests and players for each of our d behavioral aspects, it does not consider relations between these aspects.

Tensor factorization methods address this issue by representing the data in terms of local as well as of global factors [9, 24]. In this section, we first review a tensor factorization model called Relaxed Tensor Dual Decomposition into Directed Components (RTDD) or TUCKER-II decomposition [9, 27]. We then discuss a conventional as well as a new, easy-to-implement algorithm to find its factors and, finally, show how the resulting decomposition allows for game content recommendation.

RTDD genralizes Dual Tensor DEDICOM, Tensor DEDICOM, and INDSCAL [7, 24]. While the latter are confined to stacks of square similarity matrices, RTDD provides low rank representations of collections of arbitrary, bipartite tensors. Figure 2 illustrates how, given a tensor of bipartite data matrices $\mathcal{Y} = \{\boldsymbol{Y}_1, \ldots, \boldsymbol{Y}_d\}, \boldsymbol{Y}_i \in \mathbb{R}^{m \times n}$, RTDD models each slice as

$$\boldsymbol{Y}_i = \boldsymbol{A} \, \boldsymbol{W}_i \boldsymbol{B}^T. \tag{6}$$

Here, $\boldsymbol{A} \in \mathbb{R}^{m \times p}$ is the left basis matrix, $\boldsymbol{W}_i \in \mathbb{R}^{p \times q}$ is the coefficient matrix, and $\boldsymbol{B} \in \mathbb{R}^{n \times q}$ is the right basis matrix.

Given a data tensor $\mathcal{Y}$ and two user defined factorization parameters (p, q), RTDD factors can be found via minimization of a reconstruction error which is usually taken to be the following sum of squared matrix norms (also called the residual sum of squares)

$$E(\boldsymbol{A}, \boldsymbol{B}, \mathcal{W}) = \sum_{i=1}^{d} \left\| \boldsymbol{Y}_i - \boldsymbol{A} \, \boldsymbol{W}_i \boldsymbol{B}^T \right\|^2, \tag{7}$$

where $\mathcal{W} = \{\boldsymbol{W}_1, \ldots, \boldsymbol{W}_d\}$.

Seen as a function of a product of three variables $\boldsymbol{A}$, $\boldsymbol{W}_i$, and $\boldsymbol{B}$, each term of this loss function is non-convex. However, fixing any two of these variables, it becomes a convex function in the remaining third variable. The minimization problem is therefore typically tackled in an alternating least squares (ALS) fashion [9, 24] where all factors are randomly initialized

and each iteration of the optimization procedure fixes two of the variables and updates the third.

Next, we present two ALS algorithms that minimize (7). For both algorithms, we will discuss details pertaining to the ALS updates of $\boldsymbol{A}$, $\boldsymbol{B}$ and each slice of $\mathcal{W}$. The first algorithm finds unconstrained factors by updating each factor using matrix regression. The second, novel algorithm forces the basis matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ to consist of orthonormal columns, which leads to computationally more efficient updates of the modified factors.

Once we have discussed both algorithms, we will present an empirical evaluation on a synthetic data set where we compare them with respect to their run time behavior and reconstruction accuracy.

6.1 Unconstrained Bipartite Tensor Factorization

Akin to other unconstrained matrix- or tensor factorization approaches [9, 10, 19, 24], we can obtain unconstrained RTDD factors that minimize (7) by considering matrix regression solutions without imposing any constraints on the optimized factors.

Similar to Tensor DEDICOM and INDSCAL [24], we obtain optimal updates for each slice $\boldsymbol{W}_i$ of $\mathcal{W}$ by computing

$$\boldsymbol{W}_i \leftarrow \boldsymbol{A}^{\dagger}\boldsymbol{Y}_p\boldsymbol{B}^{T\dagger} = \left(\boldsymbol{A}^T\boldsymbol{A}\right)^{-1}\boldsymbol{A}^T\boldsymbol{X}_i\boldsymbol{B}\left(\boldsymbol{B}^T\boldsymbol{B}\right)^{-1}. \tag{8}$$

Given this, we can derive ALS updates for unconstrained $\boldsymbol{A}$ and $\boldsymbol{B}$ by first formulating our objective in (7) in terms of single matrix norm using a block matrix representation (similar to [1, 19]) and, second, considering matrix regression updates as in (8). To update $\boldsymbol{A}$, we start by horizontally stacking the slices of the factorized data tensor $\{\boldsymbol{Y}_1, \ldots, \boldsymbol{Y}_d\}$ as well as their reconstructions (via $\boldsymbol{A}$, $\boldsymbol{B}$ and slices of $\mathcal{W}$) $\{\boldsymbol{A}\boldsymbol{W}_1\boldsymbol{B}^T, \ldots, \boldsymbol{A}\boldsymbol{W}_d\boldsymbol{B}^T\}$ as in

$$\boldsymbol{Z} = \left[\boldsymbol{Y}_1\ \boldsymbol{Y}_2 \cdots \boldsymbol{Y}_d\right] \tag{9}$$

and

$$\hat{\boldsymbol{Z}} = \left[\boldsymbol{A}\boldsymbol{W}_1\boldsymbol{B}^T\ \boldsymbol{A}\boldsymbol{W}_2\boldsymbol{B}^T \cdots \boldsymbol{A}\boldsymbol{W}_d\boldsymbol{B}^T\right]. \tag{10}$$

Considering the (linearized [24]) trace representation of (7), it is easy to show the equivalence

$$\sum_{i=1}^{d} \left\|\boldsymbol{Y}_i - \boldsymbol{A}\,\boldsymbol{W}_i\boldsymbol{B}^T\right\|^2 = \left\|\boldsymbol{Z} - \hat{\boldsymbol{Z}}\right\|^2. \tag{11}$$

Factoring $\boldsymbol{A}$ out of (10) and considering the stack matrix

$$\boldsymbol{N} = \left[\boldsymbol{W}_1\boldsymbol{B}^T\ \boldsymbol{W}_2\boldsymbol{B}^T \cdots \boldsymbol{W}_d\boldsymbol{B}^T\right], \tag{12}$$

then allows us to define a least square representation for variable $\boldsymbol{A}$, namely

$$E_{\mathrm{RTDD}}(\boldsymbol{A}) = \left\|\boldsymbol{Z} - \boldsymbol{A}\boldsymbol{N}\right\|^2. \tag{13}$$

The expression in (16) next allows fore a matrix regression update for $\boldsymbol{A}$ where

$$\boldsymbol{A} \leftarrow \boldsymbol{Z}\boldsymbol{N}^{\dagger} = \boldsymbol{Z}\,\boldsymbol{N}^T\left(\boldsymbol{N}\boldsymbol{N}^T\right)^{-1}, \tag{14}$$

which using (9) and (12) results in

$$\boldsymbol{A} \leftarrow \left(\sum_i \boldsymbol{Y}_i\boldsymbol{B}\boldsymbol{W}_i^T\right)\left(\sum_i \boldsymbol{W}_i\boldsymbol{B}^T\boldsymbol{B}\boldsymbol{W}_i^T\right)^{-1}. \tag{15}$$

Similarly, we can derive an ALS update for $\boldsymbol{B}$ if we use the fact that the sum of our matrix norm is invariant under matrix transposition

$$\sum_{i=1}^{d} \left\|\boldsymbol{Y}_i - \boldsymbol{A}\,\boldsymbol{W}_i\boldsymbol{B}^T\right\|^2 = \sum_{i=1}^{d} \left\|\boldsymbol{Y}_i^T - \boldsymbol{B}\,\boldsymbol{W}_i^T\boldsymbol{A}^T\right\|^2. \tag{16}$$

Stacking $\{\boldsymbol{Y}_1^T, \ldots, \boldsymbol{Y}_d^T\}$ and $\{\boldsymbol{B}\boldsymbol{W}_1^T\boldsymbol{A}^T, \ldots, \boldsymbol{B}\boldsymbol{W}_d^T\boldsymbol{A}^T\}$ as in (9) and (10) followed by factoring out $\boldsymbol{B}$ as in (16), then allows us to compute the following ALS update

$$\boldsymbol{B} \leftarrow \left(\sum_i \boldsymbol{Y}_i^T\boldsymbol{A}\boldsymbol{W}_i\right)\left(\sum_i \boldsymbol{W}_i^T\boldsymbol{A}^T\boldsymbol{A}\boldsymbol{W}_i\right)^{-1}. \tag{17}$$

In summary, our ALS algorithm for the factorization of a given data tensor $\mathcal{Y}$ is to randomly initialize a tensor $\mathcal{W}$ and two matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ and then to iteratively apply the updates in (8), (15), and (17) until convergence.

6.2 Incorporating Constraints for Efficiency

Next, we derive novel and more efficient update schemes for ALS based minimization of (7). The gain in efficiency is achieved by constraining the two basis matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ to be column orthonormal. That is, we require

$$\boldsymbol{A}^T\boldsymbol{A} = \boldsymbol{I}_p \tag{18}$$

$$\boldsymbol{B}^T\boldsymbol{B} = \boldsymbol{I}_q. \tag{19}$$

Similar to orthogonality constrained matrix- and tensor DEDICOM [22, 24], in case of RTDD these constraints on $\boldsymbol{A}$ and $\boldsymbol{B}$, simplify the updates of the matrices $\boldsymbol{W}_i$ in (8) to

$$\boldsymbol{W}_i \leftarrow \boldsymbol{A}^T\boldsymbol{Y}_i\boldsymbol{B}. \tag{20}$$

To derive updates for $\boldsymbol{A}$ and $\boldsymbol{B}$, we consider the (linearized [24]) trace representation of (7)

$$E = \sum_{i=1}^{d} \mathrm{tr}\left[\boldsymbol{Y}_i^T\boldsymbol{Y}_i\right] - 2\mathrm{tr}\left[\boldsymbol{Y}_i^T\boldsymbol{A}\,\boldsymbol{W}_i\boldsymbol{B}^T\right] + \mathrm{tr}\left[\boldsymbol{B}\,\boldsymbol{W}_i^T\boldsymbol{A}^T\boldsymbol{A}\,\boldsymbol{W}_i\boldsymbol{B}^T\right] \tag{21}$$

and note that the Gram matrix $\boldsymbol{Y}_i^T\boldsymbol{Y}_i$ is independent of $\boldsymbol{A}$ and $\boldsymbol{B}$. Additionally, because of orthogonality of $\boldsymbol{A}$, we have

$$\mathrm{tr}\Big[\boldsymbol{B}\,\boldsymbol{W}_i^T\boldsymbol{A}^T\boldsymbol{A}\,\boldsymbol{W}_i\,\boldsymbol{B}^T\Big] = \mathrm{tr}\Big[\boldsymbol{B}\,\boldsymbol{W}_i^T\boldsymbol{W}_i\,\boldsymbol{B}^T\Big]. \tag{22}$$

Recalling the invariance of traces under cyclic permutations, we next observe that

$$\mathrm{tr}\Big[\boldsymbol{B}\,\boldsymbol{W}_i^T\boldsymbol{W}_i\,\boldsymbol{B}^T\Big] = \mathrm{tr}\Big[\boldsymbol{B}^T\boldsymbol{B}\,\boldsymbol{W}_i^T\boldsymbol{W}_i\Big] = \mathrm{tr}\Big[\boldsymbol{W}_i^T\boldsymbol{W}_i\Big] \tag{23}$$

which is independent of $\boldsymbol{A}$ and $\boldsymbol{B}$, too. This is then to say that the problem of *minimizing* (21) with respect to orthogonality constrained $\boldsymbol{A}$ or $\boldsymbol{B}$ is equivalent to *maximizing*

$$\hat{E} = \sum_{i=1}^{d}\ \mathrm{tr}\Big[\boldsymbol{Y}_i^T\boldsymbol{A}\,\boldsymbol{W}_i\,\boldsymbol{B}^T\Big]. \tag{24}$$

To derive an ALS update for $\boldsymbol{A}$, we can isolate $\boldsymbol{A}$ in (24) by once again resorting to cyclic permutation invariance and linearity of the trace operator, namely

$$\hat{E} = \sum_{i=1}^{d}\ \mathrm{tr}\Big[\boldsymbol{A}\boldsymbol{W}_i\,\boldsymbol{B}^T\boldsymbol{Y}_i^T\Big] = \mathrm{tr}\left[\boldsymbol{A}\sum_{i=1}^{d}\ \boldsymbol{W}_i\,\boldsymbol{B}^T\boldsymbol{Y}_i^T\right]. \tag{25}$$

randomly initialize $\boldsymbol{A}$, $\boldsymbol{B}$, and $\mathcal{W}$
// enforce $\boldsymbol{A}$ and $\boldsymbol{B}$ to be column orthogonal
$\boldsymbol{A},\boldsymbol{R} \leftarrow QR(\boldsymbol{A})$
$\boldsymbol{B},\boldsymbol{R} \leftarrow QR(\boldsymbol{B})$
while *stopping condition is not satisfied* **do**
  // update $\boldsymbol{A}$
  $\boldsymbol{D} = \sum_i \boldsymbol{W}_i\boldsymbol{B}^T\boldsymbol{Y}_i^T$
  $\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T = SVD(\boldsymbol{D}^T)$
  $\boldsymbol{A} \leftarrow \boldsymbol{U}\boldsymbol{V}^T$
  // update $\boldsymbol{B}$
  $\boldsymbol{C} = \sum_i \boldsymbol{W}_i^T\boldsymbol{A}^T\boldsymbol{Y}_i$
  $\hat{\boldsymbol{U}}\hat{\boldsymbol{\Sigma}}\hat{\boldsymbol{V}}^T = SVD(\boldsymbol{C}^T)$
  $\boldsymbol{B} \leftarrow \hat{\boldsymbol{U}}\hat{\boldsymbol{V}}^T$
  // update each slice $\boldsymbol{W}_i$ of $\mathcal{W}$
  for $i \in \{1,2,\ldots,d\}$ **do**
    $\boldsymbol{W}_i \leftarrow \boldsymbol{A}^T\boldsymbol{Y}_i\boldsymbol{B}$

Alg. 1: Alternating least square algorithm to find optimal RTDD factors under the constraint that basis matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ are orthogonal. The algorithm starts with randomly initialized factors, orthogonalizes the basis matrices, and then modifies each factor using the updates derived in the text until a predefined stopping condition is reached.

Introducing $\boldsymbol{D} \in \mathbb{R}^{p\times m}$ where $\boldsymbol{D} = \sum_i \boldsymbol{W}_i\boldsymbol{B}^T\boldsymbol{Y}_i^T$ and considering the *thin* SVD, i.e. truncated as in (4) for $k = {*}rank(\boldsymbol{D})$), of its transpose $\boldsymbol{D}^T = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$ (where $p \ll m$), we can then reformulate (25) as

$$\hat{E} = \mathrm{tr}\Big[\boldsymbol{A}\,\boldsymbol{V}\boldsymbol{\Sigma}\boldsymbol{U}^T\Big] = \mathrm{tr}\Big[\boldsymbol{U}^T\boldsymbol{A}\,\boldsymbol{V}\boldsymbol{\Sigma}\Big]. \tag{26}$$

Note the reformulation in (26) results in a trace of a product of a semi-orthonormal matrix (that can be turned into an orthogonal matrix by padding rows and/or columns) $\boldsymbol{U}^T\boldsymbol{A}\,\boldsymbol{V}$ and a non-negative diagonal matrix $\boldsymbol{\Sigma}$. According to an important yet lesser known result [26], such traces are upper bounded by $\mathrm{tr}[\boldsymbol{\Sigma}]$. Using that the maximum of the expression in (24) is attained for $\boldsymbol{U}^T\boldsymbol{A}\,\boldsymbol{V} = \boldsymbol{I}_p$ and that $\boldsymbol{U}^T\boldsymbol{U}\,\boldsymbol{V}^T\boldsymbol{V} = \boldsymbol{I}_p$, allow us to define an update for $\boldsymbol{A}$ that minimizes (7), namely

$$\boldsymbol{A} \leftarrow \boldsymbol{U}\,\boldsymbol{V}^T. \tag{27}$$

Following similar reasoning, we can also derive an ALS update for $\boldsymbol{B}$. That is, introducing $\boldsymbol{C} = \sum_i \boldsymbol{W}_i^T\boldsymbol{A}^T\boldsymbol{Y}_i$ and considering the SVD of its transpose $\boldsymbol{C}^T = \hat{\boldsymbol{U}}\hat{\boldsymbol{\Sigma}}\hat{\boldsymbol{V}}^T$ leads to

$$\boldsymbol{B} \leftarrow \hat{\boldsymbol{U}}\,\hat{\boldsymbol{V}}^T. \tag{28}$$

Algorithm 1 summarizes our novel ALS algorithm for computing orthogonality constrained RTDD factors. It randomly initializes $\boldsymbol{A}$, $\boldsymbol{B}$, and $\mathcal{W}$ and modifies each factor using the updates in (20), (27), and (28) until a predefined stopping condition is reached. It is worth mentioning that, since the initial values of $\boldsymbol{A}$ and $\boldsymbol{B}$ have to be column orthogonal, we can project the randomly initialized matrices to contain column orthonormal vectors using their QR decompositions[2] and set the resulting basis matrices to represent the initial values of $\boldsymbol{A}$ and $\boldsymbol{B}$.

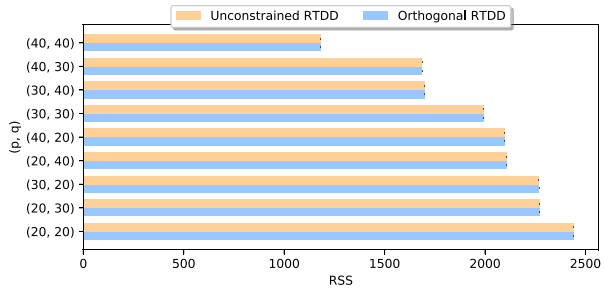
6.3 Comparative Evaluation on a Synthetic Dataset

In order to compare the two algorithms we discussed in the previous section, we next present an empirical evaluation on a synthetic dataset. In particular, we will compare their run times and their reconstruction errors (7).

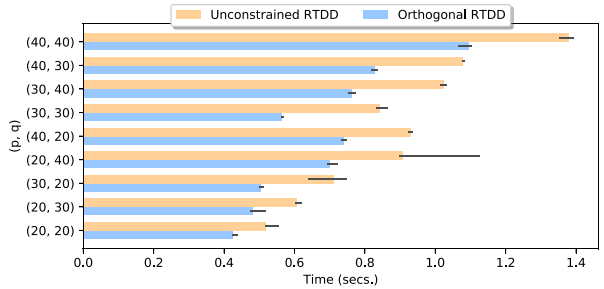
We constructed a random tensor $\mathcal{Y} \in \mathbb{R}^{40\times 50\times 200}$ with 10% sparsity to simulate the sparsity of real world settings [1, 21] and factorized it using our algorithms considering the same initial factors (with orthogonal $\boldsymbol{A}$ and $\boldsymbol{B}$ and unconstrained $\mathcal{W}$ as in Algorithm 1), parameter sets (all possible combinations of (p, q) over $\{20, 30, 40\}$) and stopping conditions. The latter is set to either having a reconstruction error difference of less than 10^{-6} or reaching 100 ALS iterations (for the outer loop of e.g. Algorithm 1).

We ran both the conventional unconstrained and the novel orthogonality constrained algorithm 30 times and report average run times and reconstruction errors (together with respective minimum and maximum values) in Fig. 3. Given these results, we observe that our algorithm for computing

[2] The QR decomposition of an arbitrary matrix $\boldsymbol{M}$ is to compute $\boldsymbol{Q},\boldsymbol{R} \leftarrow QR(\boldsymbol{M})$, where $\boldsymbol{Q}$ is orthogonal and $\boldsymbol{R}$ is upper triangular.



(a) reconstruction error



(b) run time

Fig. 3 Empirical evaluation of **a** run time and **b** accuracy [according to (7)] of our unconstrained and orthogonality constrained RTDD algorithms. Whiskers below and above the mean values indicate observed minimum and maximum values. These results show that, although it reaches almost identical reconstruction errors, the idea of constraining the basis matrices to orthogonality speeds up computation by up to 23%

constrained RTDD factors always runs faster (with an average speed up of 23%) than its uncojstrained counterpart. At the same time, it reaches almost identical reconstruction errors. We thus conclude that Algorithm 1 is preferable for practical applications.

6.4 Quest Recommendation with Factorized Tensors

Considering our quest data tensor $\mathcal{X}$ from the previous section, we note that, once we have found optimal RTDD factors $\{\boldsymbol{A}, \boldsymbol{B}, \mathcal{W}\}$ using one of our ALS algorithms, we can construct a player factor tensor $\hat{\mathcal{P}} = \{\hat{\boldsymbol{P}}_1, \ldots, \hat{\boldsymbol{P}}_d\}$ with slices $\hat{\boldsymbol{P}}_c = \boldsymbol{W}_c \boldsymbol{B}^T$. Similar to the NO and MF approaches, our RTDD approach is based on finding similar players applying the composite similarity measure (3) to $\hat{\mathcal{P}}$.

7 Online Evaluation of Our Recommenders

In this section, we present results from practical evaluations of the different quest recommender techniques we discussed above. All our results were obtained "in the wild", i.e. from analyzing online player data recorded while the game was operational.

7.1 Settings

We first conducted a pilot study with a small number of players to test our platform. Results were used to fix bugs and to tweak the original handcrafted content-based recommender. We then evaluated the above collaborative filtering methods on data obtained from a total of 243,279 players, 129,983 of which were of interest because they played the game beyond its tutorial stage. Among these players, we randomly sampled a cohort of 25,686 players (a fifth of the total population) to benchmark our recommender approaches via bucket testing.

Our test subjects played the game on 12 different operating systems and three types of platforms including phones, tablets, and desktop computers. The total number of playing sessions on the different platforms were 444,882, 293,669, and 63,403 respectively. Subjects were randomly assigned to nine different groups (consisting of 2854 players each) so that we could evaluate the different quest recommender approaches discussed above, namely: random quest assignment (baseline), content based recommendation (CB), neighborhood oriented recommendation (NO), matrix- and tensor factorization based recommendation (MF and RTDD, respectively). Each of our data driven recommender methods used the same weights for the similarity calculation in (3). We did set them based on initial play-testing results as every game aspect has a different impact on the gameplay: weights for game aspects regarding level and accuracy were set to 0.25, whereas the weight for health loss was 0.15. The game aspects strength, agility, and damage were weighted by 0.1 and, finally, the weight corresponding to completion time was set to 0.05.

As the number of basis vectors in a factor model crucially impacts recommendation quality [3, 15, 20], we considered three different settings for both for MF and RTDD. Note that, unlike our CB and NO methods, our MF and RTDD models required us to compute and maintain the factorized matrices to support the filtering process. However, since we are dealing with dynamically changing data, the more frequent these models are updated, the better the results will be. On the other hand, recomputing these models too frequently can strain the game server. We therefore tested different update cycles and found that a factorization run every three hours yields a reasonable trade-off between capturing different temporal behavior and computational demands.

Since our empirical evaluations in Sect. 6.3 showed that constraining the basis matrices of RTDD to be orthogonal can crucially improve run time while

Table 1 Daily retention rates for the first week of game play (higher values are better). The recommendation algorithms we compared include random (baseline), content-based (CB), neighborhood oriented (NO), and, matrix (MF) and tensor factorization (RTDD) based models

	Baseline	CB	NO	MF	RTDD
Day 1	8.21	8.64	7.79	8.92	8.93
Day 2	5.62	6.62	5.90	6.49	6.78
Day 3	3.96	4.64	4.40	4.55	4.74
Day 4	2.88	3.49	2.89	3.30	3.48
Day 5	2.02	2.44	1.82	1.94	2.30
Day 6	1.19	1.55	1.11	1.25	1.51
Day 7	0.25	0.54	0.61	0.47	0.72
Average	3.45	3.99	3.50	3.85	4.07

These results indicate that our tensor factorization based quest recommender consistently outperforms the other approaches and leads to the best daily average values for retaining players in the game

preserving good reconstruction accuracies, we factorized our data tensors for building quest recommenders using only Algorithm 1.

As for our evaluation metrics, we note that although these are generally game dependent, the success of F2P games usually hinges on two different yet interlinked aspects: *retention* and *monetization* [14, 21, 28]. F2P games benefit from high retention rates to increase the likelihood of in-game purchases or clicks on in-game advertisements. Since retention thus directly impacts monetization, we considered it as the primal success criterion.

7.2 Results

Table 1 shows daily average retention rates we could determine for the first week of gameplay. We note that, in general, our computational intelligence methods for quest recommendation yield higher retention rates than our baseline which achieves an average of 3.45%. Retention rates for our CB and NO approaches are 3.99% and 3.50%, respectively.

For our MF recommender, we tested different choices for the number of basis vectors, namely $k \in \{100, 500, 2000\}$, and obtained average retention rates of 3.58%, 3.85% and 3.38%, respectively. For our RTDD recommenders, we tested parameters

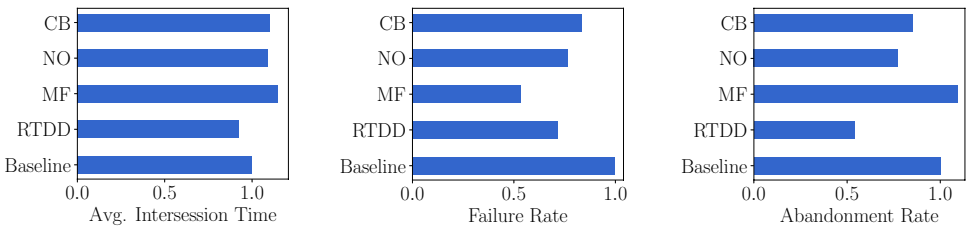
$$(p, q) \in \{(700, 500), (1000, 2000), (2000, 4000)\} \tag{29}$$

and respectively obtained retention rates of 4.07%, 3.79%, and 3.58%, which all exceeded our baseline.

Thus, the best tensor based quest recommender outperformed all other approaches and we note that this best tensor recommender ($p = 700, q = 500$) is the one with the highest data compression rate. In other words, lower dimensional factor matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ seem to capture latent relations between playing styles and the behavioral features we considered in this work. Higher dimensional factors, on the other hand, seem to lead to over fitting and thus to recommender models that do not generalize well across players.

Another retention indicator for F2P games considers the time between two consecutive sessions [6, 21] which developers aim to minimize. Analyzing average intersession times for our different groups of test subjects, we note that only those whose quests were recommended using RTDD had shorter intervals than those playing with the random baseline recommender (see Fig. 4a).

Regarding our goal of finding optimal personalized difficulty settings, we evaluated our approaches with respect to the failure rate (the ratio of failed to attempted quests) of the recommended quests. Recalling that failure rates should ideally be small as this would indicate higher levels of player satisfaction or, vice versa, lower levels of player frustration, we see in Fig. 4b that matrix based quest recommendation almost halves the failure rate of the random baseline whereas tensor based quest recommendation reduces it by 28.43%. This indicates that MF method seems to recommend less challenging quests whereas the RTDD based approach recommends quests that are challenging enough to fail yet at the same time engaging enough to retain players. This is further corroborated by our results with respect to quest



(a) time between sessions **(b)** failed quests **(c)** abandoned quests

Fig. 4 Game specific performance measures (normalized with respect to the values of our random baseline recommender) for the different recommender approaches tested in our evaluation: **a** average time (in hours) between players' play sessions, **b** ratio of failed to attempted quests, and **c** ratio of abandoned to attempted quests. For each of these measures, lower values are better

abandonment rates in Fig. 4c. These are again lowest for the RTDD recommender which improves on the baseline by 46% whereas the MF method leads to even more abandoned quests than the baseline. This suggests that RTDD based quest recommendation provides a good match between player profiles and preferences.

We finally note that any user study may suffer from possible nondeterminism of the results. Hence, in order to verify whether our results were reproducible, we conducted a second, independent study with a cohort of 10,760 unique players and observed that our results were consistent.

8 Conclusion and Future Work

Concerned with the problem of in-game content recommendation for increased player retention, we described, implemented, and evaluated several recommendation algorithms. In particular, we considered the commercial F2P game *Trollhunters: Adventures in the Troll Caves* and specifically addressed the problem of recommending quests that match players' skills and preferences.

Our main algorithmic contribution was a novel tensor factorization approach to collaborative filtering. We compared it against several baseline techniques using an online bucket testing evaluation based on extensive real world data. Our results showed the tensor factorization based approach to outperform the other methods and to yield notably better performance than an intricate, handcrafted rule-based system previously employed in the game. In contrast to the latter, our automatic approach does not require expert knowledge but works in a purely data driven manner. All in all these results demonstrate the viability of advanced game analytics methods even for smaller titles.

In future work, we will adapt the collaborative filtering approach presented in this paper to other kinds of game content as well as to other genres of games.

Acknowledgements We would like to thank the anonymous reviewers for their insightful comments. We would like to thank Flying Sheep Studios and the developers of *Trolljäger: Abenteuer in den Trollhöhlen* for creating the platform, providing us with access to their analytics suite, and supporting us with the evaluation process. Additionally, we would like to thank SRTL for supporting us to conduct this study. In parts, the work reported here was funded by the Fraunhofer Center for Machine Learning within the Fraunhofer Cluster of Excellence Cognitive Internet Technologies (CCIT).

References

1. Bader B, Harshman R, Kolda T (2007) Temporal analysis of semantic graphs using ASALSAN. In: Proceedings of the IEEE international conference on data mining (ICDM), pp 33–42
2. Chen J (2007) Flow in games (and everything else). Commun ACM 50(4):31–34
3. Cremonesi P, Koren Y, Turrin R (2010) Performance of recommender algorithms on top-*N* recommendation tasks. In: Proceedings of the ACM conference series on recommender systems (RECSYS), pp 39–46
4. Drachen A, Thurau C, Togelius J, Yannakakis GN, Bauckhage C (2013) Game data mining. In: Seif El-Nasr M, Drachen A, Canossa A (eds) Game analytics—maximizing the value of player data. Springer, Berlin, pp 205–253
5. Furnas GW, Deerwester S, Dumais ST, Landauer TK, Harshman RA, Streeter LA, Lochbaum KE (1988) Information retrieval using a singular value decomposition model of latent semantic structure. In: Proceedings of ACM SIGIR conference on research & development in information retrieval (SIGIR), pp 465–480
6. Hadiji F, Sifa R, Drachen A, Thurau C, Kersting K, Bauckhage C (2014) Predicting player churn in the wild. In: Proceedings of the IEEE conference on computational intelligence and games (CIG)
7. Harshman RA (1978) Models for analysis of asymmetrical relationships among N objects or stimuli. In: Proceedings of joint meeting of the psychometric society and the society for mathematical psychology, McMaster University, Hamilton, Ontario
8. Hofmann T (2004) Latent semantic models for collaborative filtering. ACM Trans Inf Syst 22(1):89–115
9. Kolda TG, Bader BW (2009) Tensor decompositions and applications. SIAM Rev 51(3):455–500
10. Kroonenberg Pieter M (1994) The TUCKALS line: a suite of programs for three-way data analysis. Comput Stat Data Anal 18(1):73–96
11. Kunegis J, Schmidt S, Albayrak S, Bauckhage C, Mehlitz M (2008) Modeling collaborative similarity with the signed resistance distance kernel. In: Proceedings of the European conference on artificial intelligence (ECAI), pp 261–265
12. Leskovec J, Rajaraman A, Ullman JD (2014) Mining of massive datasets, chapter 9. Cambridge University Press, Cambridge
13. Meidl M, Lytinen S, Raison K (2014) Using game reviews to recommend games. In: Proceedings of the AAAI conference on artifical intelligence for interactive digital entertainment (AIIDE), pp 24–29
14. Runge J, Gao P, Garcin F, Faltings B (2014) Churn prediction for high-value players in casual social games. In: Proceedings of the IEEE conference on computational intelligence and games (CIG)
15. Ryan JO, Kaltman E, Hong T, Mateas M, Wardrip-Fruin N (2015) People tend to like related games. In: Proceedings of the conference on foundations of digital games (FDG)
16. Ryan JO, Kaltman E, Mateas M, Wardrip-Fruin N (2015) What we talk about when we talk about games: bottom-up game studies using natural language processing. In: Proceedings of the conference on foundations of digital games (FDG)
17. Saas A, Guitart A, Perianez A (2016) Discovering playing patterns: time series clustering of free-to-play game data. In: Proceedings of the IEEE conference on computational intelligence and games (CIG)
18. Sharma M, Ontañón S, Mehta M, Ram A (2010) Drama management and player modeling for interactive fiction games. Comput Intell 26(2):183–211
19. Sifa R (2019) Matrix and tensor factorization for profiling player behavior. LeanPub, Victoria
20. Sifa R, Bauckhage C, Drachen A (2014) Archetypal game recommender systems. In: Proceedings of Learning, Knowledge, Adaptation (LWA), pp 45–56
21. Sifa R, Hadiji F, Runge J, Drachen A, Kersting K, Bauckhage C (2015) Predicting purchase decisions in mobile free-to-play games. In: Proceedings of the AAAI conference on artifical intelligence for interactive digital entertainment (AIIDE)

22. Sifa R, Ojeda C, Bauckhage C (2015) User churn migration analysis with DEDICOM. In: Proceedings of the ACM conference series on recommender systems (RECSYS), pp 321–324
23. Sifa R, Pawlakos E, Zhai K, Haran S, Jha R, Klabjan D, Drachen A (2018) Controlling the crucible: a novel PvP recommender systems framework for destiny. In: Proceedings of the ACM Australasian computer science week multiconference (ACSW)
24. Sifa R, Srikanth S, Drachen A, Ojeda C, Bauckhage C (2016) Predicting retention in sandbox games with tensor factorization-based representation learning. In: Proceedings of the IEEE conference on computational intelligence and games (CIG)
25. Smith B, Linden G (2017) Two decades of recommender systems at Amazon.com. IEEE Internet Comput 21(3):12–18
26. Ten Berge JMF (1983) A generalization of Kristof's theorem on the trace of certain matrix products. Psychometrika 48(4):519–523
27. Tucker LR (1966) Some mathematical notes on three-mode factor analysis. Psychometrika 31(3):279–311
28. Viljanen M, Airola A, Heikkonen J, Pahikkala T (2018) Playtime measurement with survival analysis. IEEE Trans Games 10(2):128–138
29. Weber B (2015) Building a recommendation system for EverQuest landmark's marketplace. Presentation at GDC
30. Xie H, Devlin S, Kudenko D, Cowling P (2015) Predicting player disengagement and first purchase with event-frequency based data representation. In: Proceedings of the IEEE conference on computational intelligence and games (CIG), pp 230–237
31. Yanakakis GN (2012) Game AI revisited. In: Proceedings of the ACM Conference on computing frontiers (CF), pp 285–292
32. Zook A, Lee-Urban S, Drinkwater MR, Riedl MO (2012) Skill-based mission generation: a data-driven temporal player modeling approach. In: Proceedings of ACM workshop on procedural content generation in games