electronics

MDPI

Article

Aggregation of Rankings Using Metaheuristics in Recommendation Systems

Michał Bałchanowski *,† **and Urszula Boryczka** †

Institute of Computer Science, Faculty of Science and Technology, University of Silesia in Katowice, Będzińska 39, 41-200 Sosnowiec, Poland; urszula.boryczka@us.edu.pl
* Correspondence: michal.balchanowski@us.edu.pl
† These authors contributed equally to this work.

Abstract: Recommendation systems are a powerful tool that is an integral part of a great many websites. Most often, recommendations are presented in the form of a list that is generated by using various recommendation methods. Typically, however, these methods do not generate identical recommendations, and their effectiveness varies between users. In order to solve this problem, the application of aggregation techniques was suggested, the aim of which is to combine several lists into one, which, in theory, should improve the overall quality of the generated recommendations. For this reason, we suggest using the Differential Evolution algorithm, the aim of which will be to aggregate individual lists generated by the recommendation algorithms and to create a single list that will be fine-tuned to the user's preferences. Additionally, based on our previous research, we present suggestions to speed up this process.

Keywords: recommendation systems; rank aggregation; differential evolution; supervised learning; matrix factorization; metaheuristic

Citation: Bałchanowski, M.; Boryczka, U. Aggregation of Rankings Using Metaheuristics in Recommendation Systems. *Electronics* **2022**, *11*, 369. https://doi.org/10.3390/electronics11030369

Academic Editor: Stefano Ferilli

Received: 30 November 2021
Accepted: 22 January 2022
Published: 26 January 2022

1. Introduction

In today's world where the amount of information available is overwhelming for a common user, the use of systems designed to support the user in making decisions is becoming more apparent. This role is taken on by recommendation systems, which are more commonly used in various areas of our life. From buying items on auction sites through selecting a movie to adding new friends on social networks. The growing popularity of this type of website means that there is a real demand for recommendation systems that work efficiently and not only increase the quality of the generated recommendations but also ensure their novelty and diversity [1].

Within the recommendation systems, we can distinguish two main approaches to creating a recommendation. They can be based on an attempt to predict what rating (e.g., on a scale from 1 to 5) the user would give to an item in the system. They can also attempt to predict a certain set of items, most often presented in the form of a list that would be recommended to the user [2] (this problem is also called the top-N recommendations problem). Additionally, we can rely on data entered directly by the user or we can infer their preferences by observing how they use the system.

This article will also discuss the problem of rank aggregation, which has been described thoroughly in the literature, especially in the context of information retrieval systems [3–5] and proven to be NP-hard [6] even for small collections of ranks (e.g., 4 or more). However, according to some researchers [7], this topic has not yet been sufficiently studied in the context of recommended systems. Depending on the dataset used, individual recommendation algorithms can generate different recommendations, and choosing one particular algorithm over others can decrease the quality of recommendations for some of the users.

Therefore, the use of aggregation techniques has been proposed also in this context where the aim is to combine the individual lists generated by different recommendation techniques in order to create one "super" list.

Additionally, due to the fact that we will be optimizing the average precision (AP) measure, the Differential Evolution (DE) algorithm will be used, which is a metaheuristic that makes the direct optimization of this measure possible [8]. Our method is universal, and thus any metaheuristic algorithm that is used for real-valued optimization can be used here (e.g., PSO [9]). We chose the DE to conduct our research, due to the fact that it is well-suited for this type of optimization [10–12]. DE is arguably one of the most versatile and stable population-based search algorithms that exhibits robustness to many different optimization problems [13]. Additionally, it is relatively simple to implement and has a small number of control parameters, which makes this algorithm easy to tune.

The main contribution of this paper is to present how the DE algorithm can be applied to the problem of rank aggregation in recommendation systems, which will be supported by tests performed on the MovieLens 100k data set [14]. We will also present, based on our previous work [15], how to accelerate this algorithm while generating ranking lists of items using a dedicated fitness function. This function can also be successfully used in other metaheuristics that use real-valued representations of individuals in a population. In addition, we will present research that will show that the use of metaheuristic algorithms in the context of the problem of rank aggregation can be additionally justified due to the resistance of these techniques to algorithms that generate low-quality recommendations.

The article is divided into six chapters. Section 2 constitutes a literature review with information about the current literature. Section 3 presents a formal definition of a recommendation system, an explanation of the ranking aggregation problem and the Differential Evolution algorithm. Section 4 presents a description of our algorithm along with the system architecture and a figure showing a simple example regarding how the matrix fitness function is calculated. Section 5 discusses how the test environment was prepared for conducting the experiments and presents the results with commentary. The final Section 6 discusses our conclusions and research proposals for the future.

2. Literature Overview

The problem of recommendations can be presented as the problem of predicting how a user would rate a given item (e.g., on a scale from 1 to 5) [16], or as the problem of creating a list of suggested items and is referred to as the Top-N recommendation problem [17]. In fact, the latter is more similar to the real-life scenario when working with recommendation systems [18], where the recommendations are most often presented in the form of a list of suggested items in which the elements at the beginning are more important than the ones at the end.

There have been many works describing this approach in the context of recommendation systems [2,17]. In order to evaluate the quality of such recommended lists, measures that take into account the order in which the items appear on the list are used, e.g., Mean Average Precision (MAP) and Normalized Discounted Cumulative Gain (NDCG). Due to the fact that these measures are usually difficult to directly optimize, metaheuristic algorithms can be applied here [8,19]. A good review of evolutionary algorithms in recommendation systems is the paper [20], in which the authors presented an overview of the current research in this area and suggestions for research in the future.

In this article, we also pay attention to the problem of rank aggregation. A great deal of work has been done on this subject, especially in the context of information retrieval systems [21]. We generally divide the algorithms used for rank aggregation into two categories: permutation-based and score-based. There are many suggested techniques in the literature, for example: Borda Count [6], COMB* [22] (e.g., COMBSUM and COMBMNZ), or OutRank [23]. Within the context of recommendation systems, there have also been several works addressing this problem. In [24], a system for creating recommendations for the entire group of users was suggested, instead of as usually done for one user only.

In the work [25], the authors suggested creating a multi-criteria recommendation system, which, in addition to the quality of the generated recommendations, also took into account measures, such as novelty and diversity. In [26], the authors used genetic programming to create a recommendation system that generated recommendations by optimizing the MAP measure. It is also worth paying attention to [7], in which the researchers asked themselves whether the problem of rank aggregation in the context of recommendation systems is worth looking into. They performed extensive experiments and suggested the direction in which future work in this area should go.

3. Background of the Research

This chapter explains the basic information and the definitions used in this article. At first, the definition of the recommendation system, methods of obtaining feedback from users and the problem of matrix factorization will be discussed. Then, we will present the problem of rank aggregation in the context of recommendation systems. Finally, we will present a metaheuristic algorithm that will be used during our research.

3.1. Recommender System

In a recommendation system, we distinguish a certain set of users $U = u_1, \ldots, u_{|U|}$ and a certain set of items $I = i_1, \ldots, i_{|I|}$. Each of the users $u \in U$ has interacted with some of the items $i \in I$. The task of the recommendation system for the Top-N recommendation problem is to, on the basis of the historical data collected in the system, predict the user's next choices and create a list of items that are likely to interest the user. High-quality recommendations contribute to user satisfaction, which can translate into an overall good impression when using the platform. Depending on what kind of feedback is obtained from the user, recommendation techniques can be based on data from:

- Implicit feedback—This feedback is obtained by analyzing the user's behavior in the system, e.g., clicking on a specific product, page views and adding an item to the basket [27]. This type of feedback is easier to obtain as there is no need to ask the user to interact with the system (e.g., commenting and rating items). The main disadvantage of this approach is the lack of information on whether the interaction with the object was positive or negative [28]. For example, the user may have accidentally added an item to the basket and later removed it, and the mere fact of opening a page does not mean that the user likes the item. For this reason, the implementation of systems based on this type of data is associated with a number of challenges and has been described in many works [29,30].
- Explicit feedback—Feedback is obtained from the user in a direct way, for example the system asks the user to rate a given item [31]. The main advantage of this type of feedback is that it is easier to determine whether the interaction with the system was positive or negative. For example, if the user can enter a rating on a scale from 1 to 5 and selects a rating of 5, then, with a high probability, it can be assumed that this is an item that the user likes.
- Hybrid feedback—This is a combination of the two previously discussed techniques [32].

It should also be noted that recommendation systems often do not have good quality features for users and items. For this reason, various methods of obtaining them have been proposed, and one of the most popular techniques is to factorize the user–item matrix. With this, we can obtain features that are also called latent features. More on the subject can be found in [33].

3.2. Rank Aggregation Problem

This section describes the problem of rank aggregation in the context of recommendation systems. We define a ranking as an ordered list of items $\tau = [i_j >= i_h >= \cdots >= i_z]$, where the items at the beginning of the list (first position) are more significant than those at the end (last position). Item positions i_j in ranking τ, we define as $\tau(i_j)$. Two items $i_j \in \tau$

and $i_h \in \tau$ can be compared by checking their position in the list τ. If the item i_j is ranked higher in the τ in comparison to the item i_h, it is defined as $\tau(i_j) > \tau(i_h)$.

In recommendation systems, aggregations are generated through various algorithms, where a single algorithm will be defined as a_h, and a set of n recommendation algorithms will be defined as $A = \{a_1, a_2, \ldots, a_n\}$. Each of the algorithms $a_h \in A$ generates a ranking τ, and the set of all n created rankings is defined as $T = \{\tau^1, \tau^2, \ldots, \tau^n\}$. In addition, all algorithms that generate recommendations take, as input, matrix $M_{m \times n}$. Each row in this matrix represents a user $u_i \in U$, and each column represents an item $i_j \in I$. The value of this matrix $M_{i,j}$ corresponds to the rating given by the user u_i to the item i_j. Note that users rate only a small fraction of the items appearing in such a matrix; therefore, such a matrix is very sparse.

The problem of rank aggregation can be defined as the problem of finding such a combination of rankings in T generated by a set of recommendation algorithms A for each user $u_i \in U$, to create a single list ("super-list") that will optimize a given criterion (in our case, the average precision) to the greatest extent. Such a list should, in theory, be "better" than individual lists.

3.3. Differential Evolution

In order to optimize the AP measure, the Differential Evolution algorithm was used, which is a metaheuristic developed by K. Price and R. Storn [10]. It is based on individuals, which are represented as vectors of real numbers. For this reason, it is primarily suitable for the optimization of continuous functions, although there are papers that have suggested modifications to the algorithm and its adaptation to the optimization of discrete problems [30].

There is a population P of individuals, where each individual is a solution to an optimization problem, often represented as a d dimensional vector of real-valued numbers. The initial population P can be initialized randomly and should cover the entire search space. In the classic version of the algorithm, this is assumed to have a uniform probability distribution. In order to determine how good a given individual is in the population, it is necessary to define the fitness function, which assigns a certain value to each individual in the population.

This value is later used in the selection process, which is the process of choosing which individuals should go to the next generation. With each iteration, the algorithm attempts to improve the population of individuals until the stopping criterion is reached (e.g., a certain number of iterations). Owing to the use of crossover and mutation operators [34], the population of individuals changes and the algorithm attempts to find a better solution. Mutation creates a new individual by combining three randomly selected individuals and can be expressed with the following formula:

$$\vec{v_i} = \vec{x_{r_1}} + F(\vec{x_{r_2}} - \vec{x_{r_3}}), \tag{1}$$

where r_1, r_2 and r_3 are random unique individuals ($r_1 \neq r_2 \neq r_3$). The F parameter is the parameter responsible for amplification and usually takes a value in the range $[0, 1]$. After creating a new individual $\vec{v_i}$ using the mutation operator, we use the crossover operator according to Formula (2). The CR parameter is the parameter that determines the crossover probability. Additionally, there is a *rand* function that generates a random number between [0, 1].

$$u_{i,j} = \begin{cases} v_{i,j} & \text{if } (rand(j) \leq CR \text{ or } i = i_{rand}) \\ x_{i,j} & otherwise. \end{cases} \tag{2}$$

4. Suggested AggRankDE Method

Our AggRankDE method is designed based on the values issued by the individual recommender algorithms for each item i in the set of all items I to find a vector of the weight W that achieves the largest AP value on the training set TS. It should be noted

that this vector is created for each user $u_i \in U$ separately, since each user has their own individual recommendation preferences. Additionally, based on our previous research, we suggest a matrix representation for the scores given by individual algorithms and the population of individuals of the DE algorithm.

Details of this representation can be found in our previous work [15], and a simple example is presented in Figure 1. As a result it is easier to parallelize the process of learning user preferences and, thus, to reduce the computation time that is needed to find the particular preference vector W.

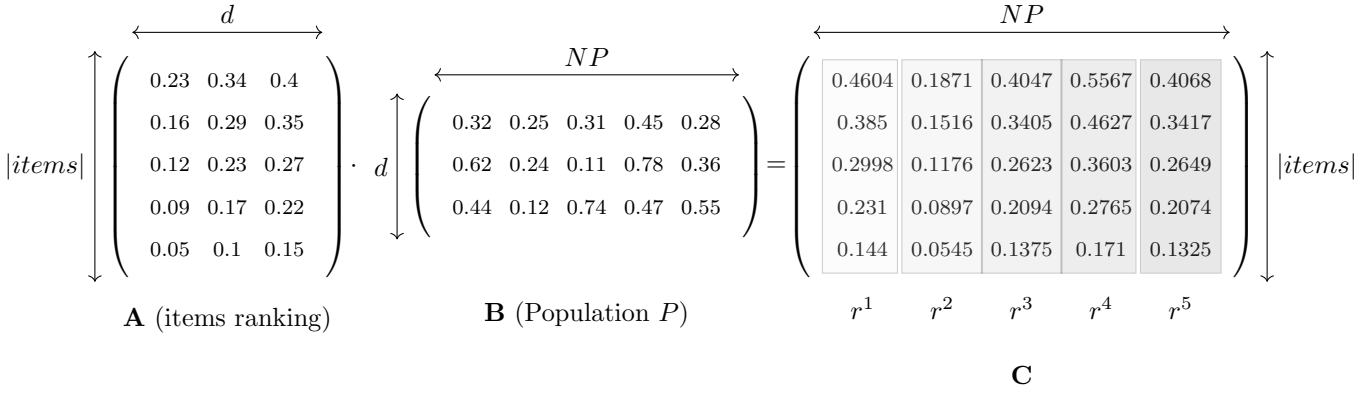

Figure 1. Toy example of the multiplication of two matrices. Matrix A represents scores assigned by the recommendation algorithms to each item $i \in I$ and some population P (real value vectors) of the metaheuristic algorithm represented by matrix B. Matrix product C represents new scores for each item $i \in I$, which, after sorting, create new rankings τ^n where $n \in \{1, 2, \ldots, NP\}$.

The hybridization technique was taken from [25] and is based on assigning weights $W = \{w_{a_1}, w_{a_2}, \ldots, w_{a_n}\}$ for each algorithm a_h, from the set of algorithms $A = \{a_1, a_2, \ldots, a_n\}$. The aggregated value for each item is calculated according to the formula:

$$\hat{p}(i_j|u_i) = \sum_{h=1}^{n} \hat{p}_{a_h}(i_j|u_i)\, w_{a_h} \tag{3}$$

where w_{a_h} is the weight assigned to the algorithm $a_h \in A$, with each algorithm assigning a value of $\hat{p}_{a_h}(i_j|u_i)$ to each item i_j, which determines the degree of potential interest of user u_i in this item. We should also remember to use the normalization technique so that all the algorithms in A can operate on the same scale.

The use of the metaheuristic algorithm based on evolution is associated with the need to define the fitness function so that, in subsequent iterations, the algorithm can reward individuals who are better adapted, i.e., with a greater value of the fitness function. In our case, this will be the average precision (AP) measure calculated for the active user u_A as follows:

$$Fitness = AP@k(R, S) \tag{4}$$

where S is the set of items recommended by the system and R is the set of items that user u_A rated in TS. According to our experiments, the value of k in AP during the learning process should be defined as the number of items that the user u_A rated in his TS. In our opinion, such a value is most appropriate due to the fact that it does not cause the algorithm to overfit. The details for how to calculate AP, especially in the context of recommendation systems, can be found in our paper [35]. The architecture of our system is presented below Figure 2.

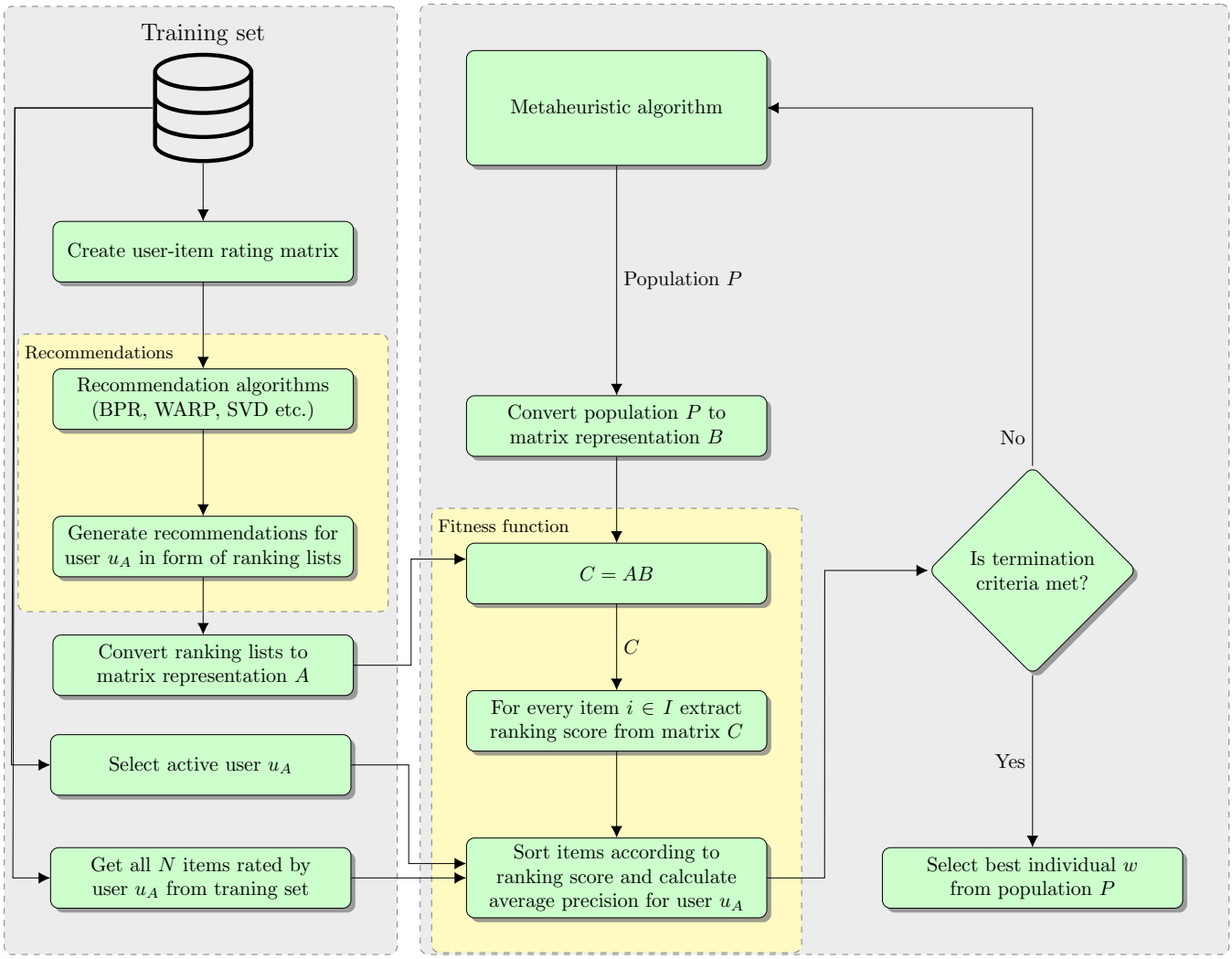

Figure 2. System architecture. The recommendation process is divided into two phases. In the first phase, recommendation algorithms generate recommendations in the form of lists, and active user u_A is selected with all his N items from the training set. In the second phase, a metaheuristic algorithm works (in our case DE) with the dedicated fitness function, which allows for faster calculation of item scores, on the basis of which, new rankings will be created.

5. Experimental Evaluation

Due to the fact that recommendations are most often presented to users in the form of a list, in our experiments, we used the average precision measure (AP) and the mean average precision measure (MAP). The AP measure is used in the context of a specific (one) user, and, in our research, it was used to compare the list of items recommended to the user with the list of items available in the test set for a given user. This allowed us to calculate the quality of the generated recommendations.

In addition, it should be noted that this measure also takes into account where the relevant items are located on the list. If the relevant items are higher (closer to the first position), then the AP value is also higher. Due to the fact that metaheuristics are computationally expensive, we chose only a certain subset of users for the experiments. We randomly selected 50 users who rated at least 150 movies in the dataset. The experiments carried out as part of this paper were performed using the popular MovieLens 100k dataset. The AggRankDE algorithm adopts four algorithms as the input: SVD, WMF, BPR and WARP. All of them are based on matrix factorization, and thus features are generated for each item and for each user on the basis of the user–item matrix.

These features are called latent features due to the fact that their meaning cannot be explained. In addition, these algorithms are considered to be the current state-of-the-art and are often used to compare research results in recommendation systems for the Top-N recommendation problem. The research environment was implemented in Python and C#, and the research was carried out on a computer with an Intel Core i5-7600 (3.50 GHz) with 16 GB RAM.

5.1. Parameters Tuning

Before creating an aggregation, the parameters of the algorithms that are included must be tuned. To this end, experiments were conducted to tune their values so that they could achieve the best possible MAP measure on the set of users used for the experiments. This is an important step, due to the fact that improper tuning of the parameters can result in the generation of poor quality recommendations. Table 1, presented below, shows the parameter values used during the tuning process.

This process consisted of first setting all parameters to the default values and then changing only one parameter that was selected for the tuning. After the process was completed, the best values were saved in the ("*Best values*" column in Table 1). The detailed MAP@10 values obtained during this process for various parameters are presented in tables: Table 2 (learning rate), Table 3 (regularization) and Table 4 (latent features).

The process of tuning the CR and F parameters for the DE algorithm was also performed, and the results of these experiments are presented in Tables 5 and 6. In addition, in article [10], the authors indicated that a good value for the parameter NP is a value between $5 \cdot d$ and $10 \cdot d$, where d is the number of dimensions. The authors also point out that the parameter F, equal to 0.5, is usually a good initial value and this parameter typically takes a value in the range $[0.4, 1]$. The final values of the Differential Evolution algorithm that were used during the experiments are presented in Table 7.

Table 1. The recommendation algorithms parameters that were used during the tuning process.

Algorithm Name	Parameter Name	Values Used in Tuning Process	Default Values	Best Values
BPR	Regularization	{0.0, 0.005, 0.01, 0.05, 0.1, 0.15, 0.2}	0.0	0.0
	Learning rate	{0.005, 0.01, 0.25, 0.5, 0.1, 0.15, 0.2, 0.25, 0.3}	0.05	0.025
	Latent factors	{4, 5, 6, 7, 8, 9, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100}	10	10
WARP	Regularization	{0.0, 0.005, 0.01, 0.05, 0.1, 0.15, 0.2}	0.0	0.0
	Learning rate	{0.005, 0.01, 0.25, 0.5, 0.1, 0.15, 0.2, 0.25, 0.3}	0.05	0.15
	Latent factors	{4, 5, 6, 7, 8, 9, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100}	10	50
WMF	Latent factors	{4, 5, 6, 7, 8, 9, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100}	10	10
SVD	Latent factors	{4, 5, 6, 7, 8, 9, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100}	10	6

Table 2. Learning rate parameter tuning. This table presents MAP@10 for different parameter values. The remaining parameters are set to the default values according to Table 1.

Learning Rate	BPR_MAP	WARP_MAP
0.005	0.176	0.158
0.01	0.208	0.185
0.025	0.259	0.198
0.05	0.228	0.214
0.1	0.209	0.220
0.15	0.191	0.244
0.2	0.174	0.230
0.25	0.149	0.122
0.3	0.137	0.079

Table 3. Regularization parameter tuning. This table presents MAP@10 for different parameter values. The remaining parameters are set to the default values according to Table 1.

Regularization	BPR_MAP	WARP_MAP
0	0.228	0.214
0.005	0.170	0.173
0.01	0.171	0.147
0.05	0.170	0.073
0.1	0.166	0.002
0.15	0.166	0.005
0.2	0.009	0.000

Table 4. Latent features (dimensions) parameter tuning. This table presents MAP@10 for different parameter values. The remaining parameters are set to the default values according to Table 1.

Latent Features	BPR_MAP	WARP_MAP	WMF_MAP	SVD_MAP
4	0.20	0.21	0.19	0.18
5	0.18	0.21	0.15	0.20
6	0.17	0.21	0.20	0.21
7	0.17	0.20	0.20	0.21
8	0.19	0.21	0.20	0.19
9	0.20	0.20	0.20	0.20
10	0.23	0.21	0.22	0.19
20	0.22	0.21	0.21	0.13
30	0.21	0.25	0.18	0.08
40	0.23	0.24	0.17	0.07
50	0.22	0.27	0.17	0.06
60	0.21	0.26	0.16	0.05
70	0.21	0.25	0.13	0.06
80	0.21	0.23	0.15	0.05
90	0.21	0.24	0.14	0.04
100	0.20	0.22	0.13	0.05

Table 5. F parameter tuning. This table presents MAP@10 for different parameter values. The remaining parameters are set to the default values according to Table 1.

Amplification Factor F	DE_MAP
0.3	0.42
0.4	0.43
0.5	0.46
0.6	0.45
0.7	0.44
0.8	0.42
0.9	0.43
1	0.44

Table 6. CR parameter tuning. This table presents MAP@10 for different parameter values. The remaining parameters are set to the default values according to Table 1.

Crossover's Probability CR	DE_MAP
0.3	0.45
0.4	0.43
0.5	0.46
0.6	0.44
0.7	0.41
0.8	0.41
0.9	0.49
1	0.43

Table 7. The differential evolution parameters used in the experiments.

Parameter Name	Value
Population	50
Number of Iterations	500
Crossover's Probability	0.9
Amplification Factor F	0.5

5.2. Experimental Setup

In order to prepare the environment for testing, first, the data was prepared in an appropriate way. User ratings were sorted by the time in which a given rating was issued and then divided into two sets: training (80%) and test (20%). Owing to this approach, our algorithm attempts to predict the user's future preferences based on the user's previous activity. The task is not trivial due to the number of items from which we can choose items and which will later be presented to the user.

Fifty users were randomly selected for the study, where a recommendation was generated for each user, and then the results of the suggested recommendations were compared with the test sets of each user. The AP measure was used to calculate the quality of the generated recommendations, and then its value was averaged for all users selected for testing; thus, the tables show the results given using the MAP measure. In order to show that our algorithm gives good results, we compared it with other algorithms used for the rank aggregation problem, such as the Borda Count, Majority Judgement, Pairwise Method (Copeland's) and Score Voting (mean).

In the research, we additionally took into account the quality of recommendations that was achieved through algorithms that participated in the creation of aggregation. These included the Bayesian Personal Ranking (BPR) and Weighted Approximate-Rank Pairwise (WARP) algorithms, the implementation of which is available in the LightFM library [36]. In addition, the usual SVD algorithm marked in the results as "SVD" and a weighted matrix factorization (WMF) algorithm were implemented.

5.3. Results

In Section 5.1, we presented the process of tuning the parameters for the various algorithms used to create aggregations. This is an important step, due to the fact that the quality of the generated recommendations by the different recommendation techniques can largely depend on the parameters that are set. For example, by analyzing Table 4, it can be seen that the MAP value obtained was highly dependent on the number of latent features. Additionally, the research presented in Table 2 showed that the parameter "Learning rate", which is characteristic for the BPR and WARP techniques, also required tuning as opposed to the parameter "Regularization" (Table 3) where the default value (0) generated the best quality of the recommendations.

While analyzing the results presented in Table 8, it can be seen that the AggRankDE algorithm aggregated the recommendation algorithms and improved the overall quality of the generated recommendations even compared to other aggregation techniques. This is an important observation because it shows that one "super" list can be created from several lists to improve the quality of recommendations, which is consistent with the experimental results by [7].

Looking at the quality of the recommendations generated by the different recommendation algorithms, we can see that, depending on $MAP@$, the quality of the recommendations varies. In general, as the number of items based on which the MAP@ measure is calculated increases, it can be seen that the quality of the recommendations decreases, although the AggRankDE algorithm improved the quality of the generated recommendations in all cases.

Additionally, after the introduction of the "Random" method (Table 9), which purposefully generated poor quality recommendations, in the case of the AggRankDE, this did not significantly degrade the quality of the produced aggregation in contrast with, for

example, the Borda Count method. This indicates that the AggRankDE has some resistance to weak algorithms that are used in the aggregation.

Table 10 presents the improvement in the speed (in seconds) of the generated recommendations after implementing the matrix fitness function. Time is measured for a single user in the system and depends on the number of iterations. Looking at this table, it can be seen that the improvement in speed is significant, and this is due to the fact that the operation on entire matrices can be easily parallelized. This is particularly important in the context of metaheuristic algorithms due to the fact that computing the fitness function is the most costly step in this type of algorithm.

Table 8. The quality of the generated recommendations (MAP) for different *MAP@* values for the best parameters presented in Table 1.

MAP@	Bpr	Warp	WMF	SVD	Borda	Majority	Pairwise	Score	AggRankDE
1	0.46	0.44	0.46	0.44	0.5	0.46	0.48	0.5	0.58
2	0.39	0.36	0.43	0.37	0.4	0.38	0.39	0.41	0.43
3	0.34	0.31	0.36	0.34	0.36	0.36	0.36	0.35	0.4
4	0.32	0.28	0.28	0.32	0.32	0.32	0.33	0.33	0.36
5	0.30	0.26	0.27	0.29	0.31	0.30	0.30	0.31	0.32
6	0.27	0.25	0.24	0.26	0.28	0.28	0.28	0.28	0.29
7	0.24	0.24	0.23	0.24	0.26	0.26	0.26	0.27	0.28
8	0.23	0.23	0.22	0.23	0.24	0.24	0.25	0.25	0.26
9	0.23	0.21	0.20	0.22	0.23	0.23	0.23	0.23	0.24
10	0.22	0.21	0.20	0.21	0.22	0.23	0.21	0.21	0.24

Table 9. The quality of the generated recommendations (MAP) for different *MAP@* values for the best parameters presented in Table 1 with the additional *RANDOM* algorithm.

MAP@	Bpr	Warp	WMF	SVD	Random	Borda	Majority	Pairwise	Score	AggRankDE
1	0.46	0.44	0.46	0.44	0.04	0.26	0.42	0.5	0.2	0.56
2	0.39	0.36	0.38	0.37	0.06	0.22	0.39	0.39	0.16	0.48
3	0.34	0.31	0.34	0.34	0.01	0.19	0.34	0.36	0.13	0.42
4	0.32	0.28	0.30	0.32	0.02	0.16	0.30	0.32	0.10	0.35
5	0.30	0.26	0.29	0.29	0.02	0.15	0.28	0.29	0.09	0.32
6	0.27	0.25	0.24	0.26	0.02	0.14	0.26	0.28	0.08	0.28
7	0.24	0.24	0.24	0.24	0.02	0.14	0.23	0.26	0.07	0.27
8	0.23	0.23	0.22	0.23	0.01	0.13	0.22	0.24	0.06	0.25
9	0.23	0.21	0.21	0.22	0.01	0.12	0.21	0.23	0.06	0.24
10	0.22	0.21	0.21	0.21	0.01	0.12	0.20	0.21	0.05	0.23

Table 10. The average time (in seconds) depending on the number of iterations. The remaining parameters are according to Table 7.

Iterations	DE	AggRankDE
100	0.89 s	0.45 s
200	1.59 s	0.83 s
300	2.28 s	1.2 s
400	3.0 s	1.51 s
500	3.72 s	1.87 s
600	4.46 s	2.28 s
700	5.19 s	2.66 s
800	5.92 s	3.01 s
900	7.03 s	3.45 s
1000	7.5 s	3.86 s

When analyzing the experimental results, the application of the DE algorithm with the hybridization technique presented in [25] produced good results. However, in our paper, we suggested how to improve it by using a dedicated fitness function to directly optimize the average precision measure and to speed up its calculation process. By assigning different weights to the different algorithms included in the aggregation, the DE algorithm

optimizes the average precision measure using a weighted hybridization technique in order to obtain the highest possible value of the average precision measure on the training set.

During the testing phase, this translated into an increase in the quality of the generated recommendations. However, this process is computationally very expensive; therefore, we suggested using the matrix representation in the fitness function, which significantly accelerated the process of calculating the values for each item by the hybridization technique on the basis of which the ranking was created.

6. Conclusions

In this article, we presented how the Differential Evolution algorithm can be used to optimize the problem of rank aggregation in recommendation systems. The experiments were conducted on the database MovieLens 100k, and they showed that our algorithm improved the quality of the recommendations expressed by the MAP measure by 5% compared to other algorithms used for this purpose. Our research showed that, even using simple aggregation techniques, we could improve the quality of the generated recommendations.

In addition, in analyzing the research results, it can be seen that the AggRankDE algorithm is resistant to algorithms that generate poor-quality recommendations. We believe that this is due to the fact that, through the presence of a training phase in which the DE algorithm optimizes the AP measure, it is able to detect algorithms that generate low-quality recommendations and assign them correspondingly low weights, which results in them participating least in the creation of the list of recommended items.

Based on our previous work, we also suggested the use of matrix representation for the population of the DE algorithm and the values of coefficients calculated by individual aggregation algorithms for each item in the system. Such a representation makes it much easier to parallelize the process of calculating the values for individual items in the training phase on the basis of which new rankings (recommendations) are created. The calculation of the fitness function is the most expensive operation in the metaheuristic algorithms. In the context of the recommendation systems, this is particularly important, due to the relatively large data sets that are processed.

In following papers, we will increase the number of algorithms that are part of the aggregation, add more aggregation techniques and increase the number of data sets on the basis of which the research is carried out. We will also conduct a more detailed analysis of the effectiveness of our algorithm, taking into account a larger number of users, and conduct a more detailed analysis of how the parameters of the individual algorithms included in the aggregation and the model itself affect the quality of the generated recommendations.

Another interesting direction of research would be to take a closer look at the quality of the generated recommendations by particular algorithms in relation to individual users. Although the AggRankDE algorithm is more robust to algorithms that generate poor recommendations, the decrease in the quality is noticeable. Presumably, eliminating the weaker quality algorithms would generally improve the quality of the aggregation produced. We believe that the problem of rank aggregation within the context of the recommendation systems has not yet been sufficiently studied, and this will likely be the direction of our future work.

Author Contributions: Conceptualization, U.B. and M.B.; Formal analysis, U.B. and M.B.; Investigation, M.B.; Methodology, U.B. and M.B.; Project administration, M.B.; Software, M.B.; Validation, U.B. and M.B.; Visualization, M.B.; Writing—original draft, U.B. and M.B.; Writing—review and editing, U.B. and M.B. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data is available at https://grouplens.org/datasets/movielens/100k/ accessed on 29 November 2021.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

u	Generic user
u_i	Specific user
u_A	Active user in system for which recommendations are generated
U	The set of all users
i	Generic item
i_j	Specific item
I	Set of all items
a_h	Specific recommendation algorithm
A	Set of n recommendation algorithms $A = \{a_1, a_2, \ldots, a_n\}$
τ	Generic ranking
τ_i^r	Ranking recommended to user u_i by algorithm a_r where $r \in \{1, 2, \ldots, n\}$
$\tau(i_j)$	The position of item i_j in ranking τ
T	Set of n rankings $T = \{\tau^1, \tau^2, \ldots, \tau^n\}$
w_{a_h}	Weight assigned to recommendation algorithm a_h where $h \in \{1, 2, \ldots, n\}$
W	Set of n weights $W = \{w_{a_1}, w_{a_2}, \ldots, w_{a_n}\}$
R	Set of items that user u_A rated in his training set
S	Set of items recommended to user u_A
P	Population of metaheuristic algorithm
NP	Number of individuals in population
TS	Training set

References

1. Castells, P.; Hurley, N.; Vargas, S. *Novelty and Diversity in Recommender Systems*; Springer: Boston, MA, USA, 2015 ; pp. 881–918. [CrossRef]
2. Cremonesi, P.; Koren, Y.; Turrin, R. Performance of recommender algorithms on top-N recommendation tasks. In Proceedings of the Fourth ACM Conference on Recommender Systems, Barcelona, Spain, 26–30 September 2010; pp. 39–46. [CrossRef]
3. Dwork, C.; Naor, M.; Sivakumar, D. Rank Aggregation Revisited. 2003 . Available online: http://www.cse.msu.edu/~cse960/Papers/games/rank.pdf (accessed on 29 November 2021).
4. Vanderpooten, D.; Farah, M. An Outranking Approach for Rank Aggregation in Information Retrieval. 2007 . Available online: https://dl.acm.org/doi/10.1145/1277741.1277843 (accessed on 29 November 2021). [CrossRef]
5. Dourado, Í.C.; Pedronette, D.C.G.; da Silva Torres, R. Unsupervised Graph-based Rank Aggregation for Improved Retrieval. *CoRR* **2019** , 56, 1260–1279. [CrossRef]
6. Dwork, C.; Kumar, R.; Naor, M.; Sivakumar, D. Rank Aggregation Methods for the Web. In Proceedings of the 10th International Conference on World Wide Web WWW'01, Hong Kong, China, 1–5 May 2001; pp. 613–622. [CrossRef]
7. Oliveira, S.E.L.; Diniz, V.; Lacerda, A.; Merschmanm, L.; Pappa, G.L. Is Rank Aggregation Effective in Recommender Systems? An Experimental Analysis. *ACM Trans. Intell. Syst. Technol. (TIST)* **2020**, *11*, 16. [CrossRef]
8. Bollegala, D.; Noman, N.; Iba, H. RankDE: Learning a ranking function for information retrieval using differential evolution. In Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation, Dublin, Ireland, 12–16 July 2011; pp. 1771–1778. [CrossRef]
9. Kennedy, J.; Eberhart, R. Particle swarm optimization. In Proceedings of the ICNN'95-International Conference on Neural Networks, Perth, WA, Australia, 27 November–1 December 1995; Volume 4, pp. 1942–1948. [CrossRef]
10. Storn, R.; Price, K. Differential Evolution—A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. *J. Glob. Optim.* **1997**, *11*, 341–359. [CrossRef]
11. Bilal.; Pant, M.; Zaheer, H.; Garcia-Hernandez, L.; Abraham, A. Differential Evolution: A review of more than two decades of research. *Eng. Appl. Artif. Intell.* **2020**, *90*, 103479.
12. Ronkkonen, J.; Kukkonen, S.; Price, K. Real-parameter optimization with differential evolution. In Proceedings of the 2005 IEEE Congress on Evolutionary Computation, Edinburgh, UK, 2–5 September 2005; Volume 1, pp. 506–513. [CrossRef]
13. Feoktistov, V. *Differential Evolution*; Springer: Boston, MA, USA, 2006.
14. Harper, F.M.; Konstan, J.A. The MovieLens Datasets: History and Context. *ACM Trans. Interact. Intell. Syst.* **2015**, *5*, 19 . [CrossRef]

15. Boryczka, U.; Bałchanowski, M. Speed up Differential Evolution for ranking of items in recommendation systems. *Procedia Comput. Sci.* **2021**, *192*, 2229–2238. [CrossRef]
16. Bennett, J.; Lanning, S.; Netflix, N. The Netflix Prize. In *KDD Cup and Workshop in Conjunction with KDD*; 2007 . Available online: https://www.cs.uic.edu/~liub/KDD-cup-2007/NetflixPrize-description.pdf (accessed on 29 November 2021).
17. Deshpande, M.; Karypis, G. Item-Based Top-N Recommendation Algorithms. *ACM Trans. Inf. Syst.* **2004**, *22*, 143–177. [CrossRef]
18. Karatzoglou, A.; Baltrunas, L.; Shi, Y. Learning to rank for recommender systems. In Proceedings of the 7th ACM Conference on Recommender Systems, Hong Kong, China, 12–16 October 2013; pp. 493–494. [CrossRef]
19. Diaz-Aviles, E.; Nejdl, W.; Schmidt-Thieme, L. Swarming to Rank for Information Retrieval. In Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation GECCO'09, Montreal, QC, Canada, 8–12 July 2009; Association for Computing Machinery: New York, NY, USA, 2009; pp. 9–16. [CrossRef]
20. Horvath, T.; de Carvalho, A. Evolutionary computing in recommender systems: A review of recent research. *Nat. Comput.* **2016** , 16, 441–462. [CrossRef]
21. Klementiev, A.; Roth, D.; Small, K. *Unsupervised Rank Aggregation with Distance-Based Models ICML'08*; Association for Computing Machinery: New York, NY, USA, 2008; pp. 472–479. [CrossRef]
22. Shaw, J.A.; Fox, E.A. Combination of Multiple Searches. In Proceedings of the Second Text Retrieval Conference (TREC-2), Plainsboro, NJ, USA, 8–11 March 1994; pp. 243–252.
23. Farah, M.; Vanderpooten, D. An Outranking Approach for Rank Aggregation in Information Retrieval. In Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval SIGIR'07, Amsterdam, The Netherlands, 23–27 July 2007; Association for Computing Machinery: New York, NY, USA, 2007; pp. 591–598. [CrossRef]
24. Baltrunas, L.; Makcinskas, T.; Ricci, F. Group Recommendations with Rank Aggregation and Collaborative Filtering. In Proceedings of the Fourth ACM Conference on Recommender Systems RecSys'10, Barcelona, Spain, 26–30 September 2010; Association for Computing Machinery: New York, NY, USA, 2010; pp. 119–126. [CrossRef]
25. Ribeiro, M.T.; Ziviani, N.; Moura, E.S.D.; Hata, I.; Lacerda, A.; Veloso, A. Multiobjective Pareto-Efficient Approaches for Recommender Systems. *ACM Trans. Intell. Syst. Technol.* **2015**, *5*, 53 . [CrossRef]
26. Oliveira, S.; Diniz, V.; Lacerda, A.; Pappa, G.L. Evolutionary rank aggregation for recommender systems. In Proceedings of the 2016 IEEE Congress on Evolutionary Computation (CEC), Vancouver, BC, Canada, 24–29 July 2016; pp. 255–262. [CrossRef]
27. Oard, D.; Kim, J. Implicit Feedback for Recommender System. In Proceedings of the AAAI Workshop on Recommender Systems 2000 . Available online: https://www.aaai.org/Papers/Workshops/1998/WS-98-08/WS98-08-021.pdf (accessed on 29 November 2021).
28. Hu, Y.; Koren, Y.; Volinsky, C. Collaborative Filtering for Implicit Feedback Datasets. In Proceedings of the 2008 Eighth IEEE International Conference on Data Mining, Pisa, Italy, 15–19 December 2008; pp. 263–272. [CrossRef]
29. Rendle, S.; Freudenthaler, C.; Gantner, Z.; Schmidt-Thieme, L. BPR: Bayesian Personalized Ranking from Implicit Feedback. In Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence UAI'09, Montreal, QC, Canada, 18–21 June 2009; AUAI Press: Arlington, VA, USA, 2009; pp. 452–461.
30. Pan, R.; Zhou, Y.; Cao, B.; Liu, N.N.; Lukose, R.; Scholz, M.; Yang, Q. One-Class Collaborative Filtering. In Proceedings of the 2008 Eighth IEEE International Conference on Data Mining, Pisa, Italy, 15–19 December 2008; pp. 502–511. [CrossRef]
31. Jawaheer, G.; Szomszor, M.; Kostkova, P. Comparison of implicit and explicit feedback from an online music recommendation service. In Proceedings of the 1st International Workshop on Information Heterogeneity and Fusion in Recommender Systems, Barcelona, Spain, 26–30 September 2010; [CrossRef]
32. Chou, C.L.; Lu, T.Y. A hybrid-feedback recommender system for employment websites. *J. Ambient. Intell. Humaniz. Comput.* **2020** . Available online: https://link.springer.com/article/10.1007/s12652-020-01772-y (accessed on 29 November 2021). [CrossRef]
33. Koren, Y.; Bell, R.; Volinsky, C. Matrix Factorization Techniques for Recommender Systems. *Computer* **2009**, *42*, 30–37. [CrossRef]
34. Boryczka, U.; Juszczuk, P.; Kłosowicz, L. A Comparative Study of Various Strategies in Differential Evolution. In *Evolutionary Computing and Global Optimization KAEiOG'09*; 2009; pp. 19–26. Available online: https://www.researchgate.net/publication/230788075_A_Comparative_Study_of_Various_Strategies_in_Differential_Evolution (accessed on 29 November 2021).
35. Boryczka, U.; Bałchanowski, M. Using Differential Evolution in order to create a personalized list of recommended items. *Procedia Comput. Sci.* **2020**, *176*, 1940–1949. [CrossRef]
36. Kula, M. Metadata Embeddings for User and Item Cold-start Recommendations. In Proceedings of the 2nd Workshop on New Trends on Content-Based Recommender Systems Co-Located with 9th ACM Conference on Recommender Systems (RecSys 2015), Vienna, Austria, 16–20 September 2015; Volume 1448, pp. 14–21.