Simulated Annealing with Levy Distribution for Fast Matrix Factorization-Based Collaborative Filtering

Mostafa A. Shehata, Mohammad Nassef, Amr A. Badr
Department of Computer Science
Faculty of Computers and Information
Cairo University

Abstract—**Matrix factorization is one of the best approaches for collaborative filtering because of its high accuracy in presenting users and items latent factors. The main disadvantages of matrix factorization are its complexity, and are very hard to be parallelized, especially with very large matrices. In this paper, we introduce a new method for collaborative filtering based on Matrix Factorization by combining simulated annealing with levy distribution. By using this method, good solutions are achieved in acceptable time with low computations, compared to other methods like stochastic gradient descent, alternating least squares, and weighted non-negative matrix factorization.**

Keywords—Simulated annealing; levy distribution; matrix factorization; collaborative filtering; recommender systems; metaheuristic optimization

I. Introduction

The objective of Recommender Systems is to recommend new products or items for users based on their history [1]. There are two major approaches to create a Recommender System. The first one is the *Content Filtering* or (*Content Based*). This approach tries to create a profile for each user and item, and then tries to match these profiles [3]. The second approach is the *Collaborative Filtering*. It uses the rating history of users and items, and creates a large sparse matrix called Rating Matrix. This matrix usually contains ratings from 1 to 5. The 0's are for the incomplete ratings. The objective of the *Collaborative Filtering* is to predict these missing ratings. One of the most successful method for *Collaborative Filtering* is the *Latent Factor Model*[3]. This method tries to learn the latent features of each user and item in a fixed number of dimensions. Then represent each of them in a latent feature vector that can be used to predict the incomplete ratings or measure the similarity.

Matrix Factorization is one of the best techniques used for *Latent Factor Model*. The basic idea is to construct the *low-dimensional* matrices to approximate the original rating matrix [2], [3], [7], [12], [13].

$$R \approx U \cdot I \tag{1}$$

where $R_{M,N}$ is the rating matrix, $U_{M,K}$ is the users matrix, $I_{K,N}$ is the items matrix. M and N are the number of users and items respectively, K is the number of latent feature that represent each user and item. Where $K \ll \min(M, N)$. Row m in matrix U represents user number m in the rating matrix, whereas column n in matrix I represents item number n in the rating matrix. So, in the rating matrix, the rating of user m for item n can be calculated by the *dot product* of row m of matrix U by the column n of matrix I.

$$r_{m,n} \approx u_m \cdot i_n^T \tag{2}$$

The rating matrix is very sparse, because it contains a few users ratings. The objective of this paper is to use the known ratings to construct the *low-rank* matrices, to predict the unknown or incomplete ratings.

One of the most common evaluation metrics for *Collaborative Filtering* is *RMSE* (root mean squared error). We calculate *RMSE* only for the known rating using the following equation:

$$RMSE = \sqrt[2]{\Big(\sum_{m,n \in KR} (r_{m,n} - (u_m \cdot i_n^T))^2\Big)/|KR|} \tag{3}$$

where KR is the list of known ratings.

There is a lot of work done on Matrix Factorization and Collaborative Filtering. Here we discus three of the most popular methods.

Stochastic Gradient Descent (SGD): SGD is one of the popular Matrix Factorization methods [3]. The idea is to minimize the following cost equation:

$$\begin{aligned}\min_{u^*,i^*,b^*} \sum_{(m,n)\in KR} & (r_{m,n} - \mu - b_m - b_n - u_m \cdot i_n^T)^2 + \\ & \lambda(\|u_m\|^2 + \|i_n\|^2 + b_m^2 + b_n^2)\end{aligned} \tag{4}$$

where λ is a regularization term, μ is the overall average rating, b_m and b_n are the user and item bias, respectively.

$$e_{m,n} = r_{m,n} - \mu - b_m - b_n - u_m \cdot i_n^T \tag{5}$$

To minimize the squared-error (4), the algorithm iterates over all ratings in the training set. Then, it computes the associated prediction error in (5). Next, the error value is used to compute the gradient. The algorithm finally uses the gradient to update user bias, item bias, user matrix U, and item matrix I.

$$b_m = b_m + \gamma(e_{m,n} - \lambda b_m) \tag{6}$$

$$b_n = b_n + \gamma(e_{m,n} - \lambda b_n) \tag{7}$$

$$u_m = u_m + \gamma(e_{m,n} \cdot i_n - \lambda \cdot u_m) \tag{8}$$

$$i_n = i_n + \gamma(e_{m,n} \cdot u_m - \lambda \cdot i_n) \tag{9}$$

where γ is the learning rate. The learning rate determines the moving speed towards the optimal solution. If γ is very large, we might skip the optimal solution. If it is too small, we may need too many iterations to reach the optimal solution. So using an appropriate γ is very important.

Alternating Least Squares (ALS): ALS is very good for parallelization [11]. When we have large data, and need to distribute the computations over cluster of nodes. ALS objective is to minimize the following equation:

$$\min_{u^*,i^*,b^*} \sum_{(m,n)\in KR} (r_{m,n} - u_m \cdot i_n^T)^2 + \lambda(\|u_m\|^2 + \|i_n\|^2) \quad (10)$$

It is the same like (4), but without the bias terms. The basic idea can be summarized as follows:

1) Initialize U, and I matrices.
2) Fix I, solve for U by minimizing (10).
3) Fix U, solve for I by minimizing (10).
4) Repeat the previous two steps until converging or reaching the max iteration.

to solve the user U and item I matrices we use the following two equations, respectively:

$$u_m^T = (r_m \cdot I) \cdot (I^T \cdot I + \lambda Eye)^{-1} \quad (11)$$

$$i_n^T = (r_n^T \cdot U) \cdot (U^T \cdot U + \lambda Eye)^{-1} \quad (12)$$

where Eye is the Identity matrix.

Weighted Non-Negative Matrix Factorization (WNMF): Here we present a special type of Matrix Factorization called *Non-Negative Matrix Factorization* (NMF). The only difference is the non-negativity constraint for the input matrix, and the low rank matrices as well. The problem can be formulated as an optimization problem:

$$\begin{aligned} \min_{A,H} \quad & \|V - A \cdot H\|^2 \\ \text{subject to} \quad & A, H \geq 0 \end{aligned} \quad (13)$$

where V is the original matrix, $WandH$ are the two factorized matrices. One of the most simplest methods for NMF is *Multiplicative Update Rules* [12] [2]. It is a good compromise between speed and ease of implementation. So NMF objective (13) can be optimized using the following update rules:

$$A^{(t+1)} = A^{(t)} \frac{V \cdot H^T}{A \cdot H \cdot H^T} \quad (14)$$

$$H^{(t+1)} = H^{(t)} \frac{A^T \cdot V}{A^T \cdot A \cdot H} \quad (15)$$

The original version of *Multiplicative Update Rules* will not fit in our problem. The two rules will not be able to differentiate between the true ratings, and the incomplete ratings. So we need to modify the original rules, to be able to learn from the true ratings, then predict the incomplete. In [10] they could modify the original *Multiplicative Update Rules* to be able to do Incomplete Matrix Factorization. This method called *Weighted Non-Negative Matrix Factorization (WNMF)*. So now the new objective function is:

$$\min_{u^*,i^*} \sum_{(m,n)\in KR} (r_{m,n} - u_m \cdot i_n^T)^2 \quad (16)$$

It is similar to (4), (10), but without the bias b or regularization λ. Now we can optimize function (16) using the following two rules:

$$U^{(t+1)} = U^{(t)} \frac{(W * R) \cdot I^T}{(W * (U \cdot I)) \cdot I^T} \quad (17)$$

$$I^{(t+1)} = I^{(t)} \frac{U^T \cdot (W * R)}{U^T \cdot (W * (U \cdot I))} \quad (18)$$

where $W_{M,N}$ is a matrix which its elements are equal to 1 if the corresponding entry in R is known rating, and 0 otherwise. $(*)$ denotes to the element wise multiplication.

In this paper we focus on efficiency more than effectiveness. We assume that we have a very large data, and limited time. So we need an acceptable solution in reasonable time. So we chose the Metaheuristic algorithms for this problem, because of its ability to scape from the local optimal, and reaching good solutions in reasonable time. We used Simulated Annealing algorithm, with Levy Flight as a random walk operator.

The rest of the paper is organized as follows: in Section (II) we briefly describe the prerequisite topics that are needed before going through the proposed method. In Section (III) we describe our proposed method. In Section (IV) we discuss the experimental results, effect of each parameter, and compare our proposed method against others. Finally, Section (V) concludes the paper.

II. Preliminaries

A. *Metaheuristic Optimization*

There are two types of optimization algorithms, *Deterministic* and *Stochastic*. Deterministic algorithms usually focus on optimal solution, like Simplex method in linear programming, some of these algorithms use the derivative of the objective function, these algorithms are called *gradient based algorithms*.

In *Stochastic Optimization* we will talk about *Metaheuristic Algorithms*, we can divide *Metaheuristic* into two parts, META and HEURISTIC, META means “beyond” or “higher level”, and HEURISTIC means “to find” or “to discover by trial and error”. This type of algorithms depends on randomization and local search to find the optimal solution iteratively, whereas each iteration tries to improve the current solutions from previous iteration. Also *Metaheuristic* doesn’t guarantee the optimal solution, but it gives good quality solutions in a reasonable time. *Metaheuristic* achieves its goal by making a good balance between two major components, intensification and diversification. Intensification is to search for a better solution within the local area of the current solution. Diversification is to use the randomization to escape from the local optimum, and explore all the search space [4].

There are many types of *Metaheuristic* algorithms, like single solution, or population based, in this paper we use *Single Solution*.

B. Levy Distribution and Random Walk

We presented randomization techniques for exploring the search space (*Diversification*), local search for optimizing the current solution, and searching within the local area of it (*Intensification*). In (19) x^t is the current solution state, s is a new step or random number drawn from a probability distribution, we add s to x to move it from state t to $t+1$.

$$x^{t+1} = x^t + s \tag{19}$$

Levy Flights are a random walk that their steps are drawn from *Levy Distribution*. *Mantegna* algorithm is the best and easiest way to generate random numbers from *Levy Distribution* [4], [5], so the random walk can be achieved using the following equations:

$$x_i^{t+1} = x_i^t + \alpha L(s, \lambda), \tag{20}$$

Where α is the step size.

$$L(s, \lambda) = \frac{\lambda \Gamma \sin(\pi\lambda/2)}{\pi} \frac{1}{s^{1+\lambda}}, \tag{21}$$

$$s = \frac{U}{|V|^{1/\alpha}} \tag{22}$$

$$U\ N(0, \sigma^2), V\ N(0, 1) \tag{23}$$

Where N is a Gaussian normal distribution

$$\sigma^2 = \left[\frac{\Gamma(1+\lambda)}{\lambda\Gamma((1+\lambda)/2)} . \frac{\sin(\pi\lambda/2)}{2^{(\lambda-1)/2}} \right]^{1/\lambda} \tag{24}$$

C. Simulated Annealing

SA is one of the most popular metaheuristic algorithms. It simulates the annealing process for solids by cooling to reach the crystal state. Reaching the crystal state is like reaching the global optimum in optimization. It is a single solution algorithm. The basic idea is to perform a random walk, but with some probability called *Transition Probability* that may accept new solutions that do not improve the objective function, see (25). Accepting bad solutions with *Transition Probability* gives more exploration for the search space (*Diversification*). *Transition Probability* decreases gradually during the iterations to decrease the *Diversification* and increase the *Intensification*. This means that the algorithm will end up with accepting only better solutions [4] [6].

$$p = \exp\left[-\frac{\Delta f}{T} \right] > r \tag{25}$$

In (25) f is the difference between the two evaluation function values of the current solution and the new one. T is the current temperature which is decreased iteratively by the cooling rate. r is a random number. So the algorithm accepts bad solution if the *Transition Probability* p is greater than r.

One of the common cooling schedules is linear cooling schedule, in (26).

$$T = T_0 - \beta t \tag{26}$$

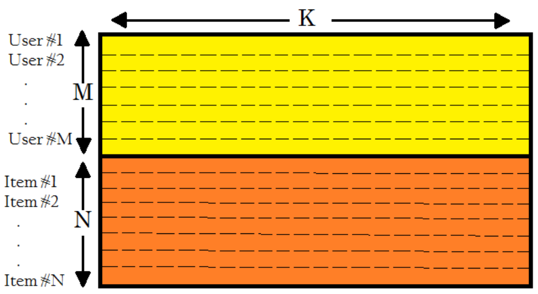

Fig. 1. Representing U matrix and I matrix in one matrix, the number of rows is equal $M + N$, and the number of columns is equal K.

T_0 is the initial temperature, β is the cooling rate, t is the pseudo time for iterations. The following pseudo code demonstrate the basic implementation of Simulated Annealing:

Algorithm 1 Simulated Annealing

```
1: Objective function $f(x), x = (x1, ..., xd)^T$
2: Initialize the initial temperature $T_0$ and initial guess $x(0)$
3: Set the final temperature $T_f$ and the max number of iterations $N$
4: Define the cooling schedule $T \mapsto \alpha T, (0 < \alpha < 1)$
5: while ($T > T_f$ and $t < N$) do
6:     Drawn $\epsilon$ from a Gaussian distribution
7:     Move randomly to a new location: $x_{t+1} = x_t + \epsilon$(random walk)
8:     Calculate $\Delta f = f_{t+1}(x_{t+1}) - f_t(x_t)$
9:     Accept the new solution if better
10:    if not improved then
11:        Generate a random number $r$
12:        Accept if $p = \exp[\Delta f/T] > r$
13:    end if
14:    Update the best $x\star$ and $f\star$
15:    $t = t + 1$
16: end while
```

III. Proposed Method

In this section we introduce our new method for solving Matrix Factorization. In our method we use *Simulated Annealing* based on *Levy Flight* as a random walk, instead of *Gaussian distribution*, see Section (II-A). We called it *SA-Levy*. We choose *Simulated Annealing* because its low computations, as it is a single solution *Metaheuristic* algorithm. So it will be very fast compared to other population based *Metaheuristic* algorithms. Also compared to the state of the art methods like *ALS*, and *WNMF Simulated Annealing* is much faster, because it needs only one matrix multiplication per iteration. Regarding *SGD*, *Simulated Annealing* is easier to be parallelized.

In *Simulated Annealing* we just need to represent the solution, and implement the evaluation function to compare the current solution against others. We use *RMSE* as an evaluation function.

Fig. 1 shows the representation of the solution, we put the two matrices users and items in one matrix to simplify the

TABLE I. SHOWS THE EFFECT OF THE NUMBER OF ITERATION ON RMSE

Iterations	5	**10**	25	50	100	200
RMSE	1.112	**1.118**	1.118	1.118	1.118	1.118

TABLE II. SHOWS THE EFFECT OF NUMBER OF LATENT FEATURES ON RMSE

Latent Features	10	**20**	30	40	50
RMSE	1.120	**1.118**	1.118	1.118	1.118

TABLE III. SHOWS THE EFFECT OF STEP SIZE OF LEVY FLIGHT ON RMSE

Step Size	0.1	**0.01**	0.001
RMSE	1.119	**1.118**	1.145

solution and the calculation [8], the number of rows is equal the number of users M plus the number of items N, and the number of columns is equal the number of latent features K.

IV. EXPERIMENTAL RESULTS

In this section we show the effect of the parameters on the *RMSE* results, we use MovieLens 1M dataset [9] in our experiments, 80% of the dataset is used for training and 20% for testing.

Tables I, II and III show how *RMSE* can be affected by the number of iteration, number of latent features, and step size, see (20). In Table I, we can see that good *RMSE* can be achieved by few number of iterations, so there is no need for many iteration to converge. In Table II we can see that best *RMSE* can be achieved starting from 20 latent features. In Table III we found that the best value for the step size is 0.01. We can say that the step size is the most important parameter in our method. It manages the balance between Intensification and Diversification, see Section (II-A). Small values of step size give more intensification, and large values give more Diversification.

Table IV shows the difference between using Gaussian distribution and Levy distribution as a random walk. Levy distribution outperform Gaussian because of its ability to escape from the local minimum [4][5].

Table V shows that SA-Levy can be outperformed by other methods in terms of effectiveness. But SA-Levy can outperform all other methods in terms efficiency, because of its low computations, where it needs only one matrix multiplication in each iteration. Unlike WNMF or ALS which need many matrix multiplications or calculating matrix inversion in each iteration. Also it is much easier than SGD to be parallelized because it doesn't need huge amount of data to be shuffled between the cluster nodes. So SA-Levy can be a good choice if we have limited time or resources and large amount of data.

TABLE IV. COMPARES LEVY AGAINST GAUSSIAN DISTRIBUTION AS A RANDOM WALK FOR SIMULATED ANNEALING

Distribution	**Levy**	Gaussian
RMSE	**1.118**	1.168

TABLE V. COMPARES SA-LEVY WITH OTHER METHODS (SGD, WNMF AND ALS)

System	SA-Levy	SGD	WNMF	ALS
RMSE	1.118	0.871	0.943	1.007

Choice of Parameters

We conducted these experiments using **Simanneal**. It is a python module for simulated annealing optimization[1], also the project source code can be found here[2]. Based on the MovieLens 1M dataset [9] we found that the best parameters are 10 Iteration, 20 latent features, 0.01 step size. For the temperature parameter we found that the best values for maximum and minimum temperature are 25000 and 2.5 respectively. To focus more on Diversification at the beginning then decrease it gradually to increase the Intensification.

V. CONCLUSION AND FUTURE WORK

We presented in this work a new method for matrix factorization based Collaborative filtering. We achieved a significant improvement in Simulated Annealing, by using Levy distribution as a random walk, instead of Gaussian distribution. We expect this contribution could fit in many optimization problems, not only matrix factorization. We think that SA-Levy is a good choice for complex matrix factorization problems. When we have a very large data, and limited time for computation. We expect that SA-Levy can be easily implemented on any distributed system, that has basic linear algebra operations, like *Apache Spark*[3], and *Hadoop*[4].

REFERENCES

[1] Kantor, Paul B. Recommender systems handbook. Eds. Francesco Ricci, Lior Rokach, and Bracha Shapira. Berlin, Germany:: Springer, 2015.

[2] Duan, Liang, et al. "Scaling up link prediction with ensembles." Proceedings of the Ninth ACM International Conference on Web Search and Data Mining. ACM, 2016.

[3] Koren, Yehuda, Robert Bell, and Chris Volinsky. "Matrix factorization techniques for recommender systems." Computer 42.8 (2009).

[4] Yang, Xin-She. Nature-inspired optimization algorithms. Elsevier, 2014.

[5] Mantegna, Rosario Nunzio. "Fast, accurate algorithm for numerical simulation of Levy stable stochastic processes." Physical Review E 49.5 (1994): 4677.

[6] Van Laarhoven, Peter JM, and Emile HL Aarts. "Simulated annealing." Simulated annealing: Theory and applications. Springer Netherlands, 1987. 7-15.

[7] Luo, Xin, et al. "An efficient non-negative matrix-factorization-based approach to collaborative filtering for recommender systems." IEEE Transactions on Industrial Informatics 10.2 (2014): 1273-1284.

[8] Salehi, Mojtaba. "Latent feature based recommender system for learning materials using genetic algorithm." Information Systems & Telecommunication (2014): 137.

[9] Harper, F. Maxwell, and Joseph A. Konstan. "The movielens datasets: History and context." ACM Transactions on Interactive Intelligent Systems (TiiS) 5.4 (2016): 19.

[1] https://github.com/perrygeo/simanneal
[2] https://github.com/mostafaashraf413/MF_SA_Levy
[3] https://spark.apache.org/
[4] http://hadoop.apache.org/

[10] Zhang, Sheng, et al. ”Learning from incomplete ratings using non-negative matrix factorization.” Proceedings of the 2006 SIAM International Conference on Data Mining. Society for Industrial and Applied Mathematics, 2006.

[11] Zhou, Yunhong, et al. ”Large-scale parallel collaborative filtering for the netflix prize.” Lecture Notes in Computer Science 5034 (2008): 337-348.

[12] Lee, Daniel D., and H. Sebastian Seung. ”Algorithms for non-negative matrix factorization.” Advances in neural information processing systems. 2001.

[13] Hernando, Antonio, Jess Bobadilla, and Fernando Ortega. ”A non negative matrix factorization for collaborative filtering recommender systems based on a Bayesian probabilistic model.” Knowledge-Based Systems 97 (2016): 188-202.