View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Waseda University Repository

A Lossless Irregular Tensor Factorization for Cross-domain Recommender System with Genetic Algorithm on Spark

A Thesis Submitted to the Department of Computer Science and Communications Engineering, the Graduate School of Fundamental Science and Engineering of Waseda University in Partial Fulfillment of the Requirements for the Degree of Master of Engineering

Submission Date: February 1st, 2019

Guodong Xue

(5117FG11-3)

Advisor: Prof. Hayato Yamana

Research guidance: Research on Parallel and Distributed Architecture

Abstract

Recommender systems analyze user behaviors and users' feedback information to recommend potentially desirable items. Users may have sufficient experience in their focused domains, however, they may lack experience in unfamiliar domains, leading to drop in accuracy of the recommender systems. Therefore cross-domains recommender systems, a system which analyzes the user information from both their familiar and unfamiliar domains, become an important emerging research topic. Weighted Irregular Tensor Factorization (WITF) was proposed as an algorithm to leverage cross-domain feedbacks across all users, which achieved higher accuracy than the algorithm which leverages feedbacks from a particular domain. WITF considers the dataset as an irregular tensor containing different items for each domain. According to the tensor theory, irregular tensor must be transferred into a regular tensor, where each domain contains the same items, before executing tensor factorization to obtain the user's latent factors. Therefore, WITF must assign a weight on each domain to ensure the minimal data loss during the transformation. We define this kind of irregular tensor factorization as "lossless irregular tensor factorization", where the "lossless" represents the minimal data loss during an irregular tensor transferring to a regular tensor. To search for the optimal domain weight configuration is a complex and time-consuming procedure. Thus, WITF simply adopts an empirical domain weight configurations. However, the performance of WITF can be optimized by searching for the optimal domain weight configuration on each domain. In this paper, we propose a model which combines Apache Spark® and genetic algorithm to search for the optimal domain weight configurations for WITF efficiently, then using the obtained optimal configuration to generate more precise users' latent factors to make recommendations. Experimental evaluation using two sizes of datasets shows 4.2% and 6.3% better accuracy in comparison with the WITF. The experiments show that our proposed method can execute on Spark platform efficiently and has the ability to search the optimal domain weights configurations of the WITF model.

Keywords --- Cross-domain Recommender Systems, Irregular Tensor Factorization, Parallel Genetic Algorithm, Apache Spark®

Contents

1. **Introduction** ... 4
  1.1 **Organization** ... 4
2. **Background** ... 6
  2.1 **Notations** ... 6
  2.2 **Cross-domain Recommender System** ... 6
  2.3 **Irregular Tensor Factorization** ... 7
  2.4 **Genetic Algorithm** ... 8
  2.5 **Apache Spark** ... 9
3. **Related Work** ... 12
  3.1 **Weighted Irregular Tensor Factorization (WITF)** ... 12
4. **Proposed Method** ... 14
  4.1 **Overview** ... 14
  4.2 **Architecture** ... 14
5. **Experimental Evaluation** ... 17
  5.1 **Datasets and Environment** ... 17
  5.2 **Implementation of Genetic Algorithm** ... 18
    5.2.1 Initial Population and Encoding ... 18
    5.2.2 Fitness Valuation ... 19
    5.2.3 Selection Operator ... 19
    5.2.4 Crossover Operator ... 20
    5.2.5 Mutation Operator ... 20
  5.3 **Parallelization Evaluation of WITF model** ... 21
    5.3.1 Parallelization Granularity of WITF model ... 21
    5.3.2 Efficiency Evaluation of WITF model on Spark Cluster ... 24
  5.4 **Evaluation of Execution Time** ... 26
  5.5 **Genetic Algorithm Generation Evaluation** ... 27
6. **Conclusion** ... 30

References ... 31

Acknowledgement ... 33

Publication ... 34

1. Introduction

With the development of the Internet, the number and variety of contents on the Internet continue to increase, and users have higher demands on searching and recommendation. Recommender systems, which makes accurate and personalized recommendations using user's difference, interest and experiences, become more and more important and have been widely used in many fields.

Collaborative filtering (CF) [1] is the most used algorithm of recommender systems and have been used in many applications, such as Amazon and Netflix. However, the data sparsity problem, such that many users do not have data in the CF recommender systems, always affects the performance of the CF recommender systems. Therefore, researchers have proposed a variety of algorithms that adopt additional data into CF recommender system. In general, users always have sufficient experience in their focused domains but lack experience in other domains. Therefore, the cross-domain recommender systems, which leverage cross-domain feedback data across all users to learn the user's latent factors, become an important research topic.

Weighted irregular tensor factorization (WITF) [2] is a model that considers cross-domain information by considering the multiple domains dataset as an irregular tensor. In this model, the weight must be assigned on each domain to ensure the minimal data loss after transferred into a regular tensor followed by processing the tensor factorization to make recommendation. However, WITF model simply adopts an empirical domain weights configurations which decide the weight by the number of items of each domain. Therefore, a method to search the optimal weights configuration for WITF model is indispensable to improve its performance.

This study mainly works on combining the WITF model with genetic algorithm on Spark to search the optimal weights configurations for making more accurate recommendations and reduce the execution time. Genetic algorithm [3] is commonly used to generate high-quality solutions for optimization and searching problems. In addition, WITF model and genetic algorithm are suitable to process on parallel. In order to speed up the computation, we adopt Spark[4] which is a powerful distributed computing platform and has been an important part of the big data analytics ecosystem.

1.1 Organization

This paper includes six sections. In section 2, we discuss the backgrounds of cross-domain recommender systems, irregular tensor factorization, genetic algorithm, and Spark. In section 3, we discuss the related work based on cross-domain recommender systems and irregular tensor factorization. In section 4, the proposed method which combines WITF and genetic

algorithm on Spark is described. In section 5, we showed the experiments and evaluation of the proposed method. Conclusions are in section 6.

2. Background

2.1 Notations

Table 1 describes notations referred to in this paper.

Table 1. Notations

Notation	Description
$\mathcal{X}$	A tensor
$\boldsymbol{X}$	A matrix (user-item rating matrix)
$\boldsymbol{X_k}$	The matrix of domain k
$\boldsymbol{X_{k,i,:}}$	A row (vector) of domain matrix $\boldsymbol{X_k}$
$\boldsymbol{X_{k,:,j}}$	A column (vector) of domain matrix $\boldsymbol{X_k}$
$\boldsymbol{X_{i,:}}$	A row (vector) of matrix $\boldsymbol{X}$
$\boldsymbol{X_{:,j}}$	A column (vector) of matrix $\boldsymbol{X}$
$\boldsymbol{U}$	User latent factor matrix
$\boldsymbol{V}$	Item latent factor matrix
$\boldsymbol{C}$	Domain latent factor matrix
$\boldsymbol{I}$	Identity matrix
$\{\omega_k\}$	A set of domain weights, $1 \le k \le \boldsymbol{K}$, $\boldsymbol{K}$ is the domain count
$\lVert\boldsymbol{X}\rVert_F$	Frobenius Norm of matrix $\boldsymbol{X}$
$\circledast$	Hadamard product

2.2 Cross-domain Recommender System

In the real-world environment, users have sufficient experience in their focused domains but lack experience in other domains. Once we used the information from users' familiar domains as auxiliary data, recommender systems could perform better in recommending potentially desirable items to the users in unfamiliar domains. Therefore, the Cross-domain recommender systems become an important emerging research topic. Cross-domain collaborative filtering (CDCF) [5] is an important research topic which focuses on product

domains. Besides the product domains, time, spatial, and other domains are also adopted to researches related to CDCF. Since matrix factorization (MF) is a widely-used model of CF recommender systems, cross-domain matrix factorization (CDMF) [6] has been proposed as an improvement technique of the CDCF method.

2.3 Irregular Tensor Factorization

In CF recommender systems, dyadic user-item relationship is considered as a core relationship. To use additional data such as tags and time, some researches adopt the tensor factorization (TF) [7] to process the triadic relationships, e.g., user-item-tag, user-item-time. Tensor factorization has two kinds of models: CP decomposition [7] and Tucker decomposition [7]. As shown in Figure 1, CP decomposition decomposes a tensor into a sum of component vectors, e.g., $\boldsymbol{U}_{i,:}$, $\boldsymbol{C}_{i,:}$ and $\boldsymbol{V}_{i,:}$, of the three latent factor matrices $\boldsymbol{U}$, $\boldsymbol{V}$ and $\boldsymbol{C}$. Compare to CP decomposition, Tucker decomposition decomposes a tensor into one core tensor $\boldsymbol{g}$ and three latent factor matrices, e.g., $\boldsymbol{U}$, $\boldsymbol{V}$ and $\boldsymbol{C}$.

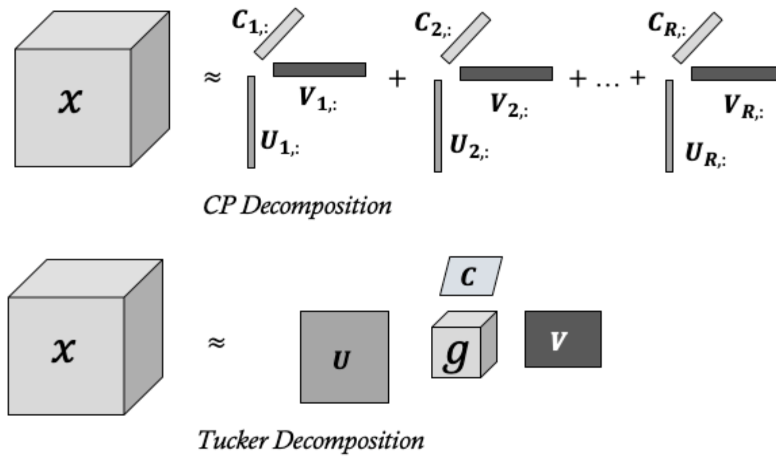

Figure 1. Two models of Tensor Factorization

To adopt the two kinds of models, the tensor must be a regular tensor that each domain contains the same items. Unfortunately, almost all datasets from the real world cannot be considered as a regular tensor with the same items in each domain. On the contrary, the datasets can be considered as an irregular tensor each of whose domains contains specific items. As shown in Figure 2, it is necessary to transfer the irregular tensor into the regular tensor which has the same set of virtual items. However, during the transfer, it is inevitable to have a data loss [2].

Therefore, we must ensure the data loss is minimal since we expect to utilize the origin data

from the irregular tensor as much as possible, and then obtain the most accurate latent factors by processing factorization for the regular tensor which transferred from the irregular tensor. We define this kind of irregular tensor factorization as the "lossless irregular tensor factorization", where the "lossless" represents the data loss is minimum during an irregular tensor transferring to a regular tensor.

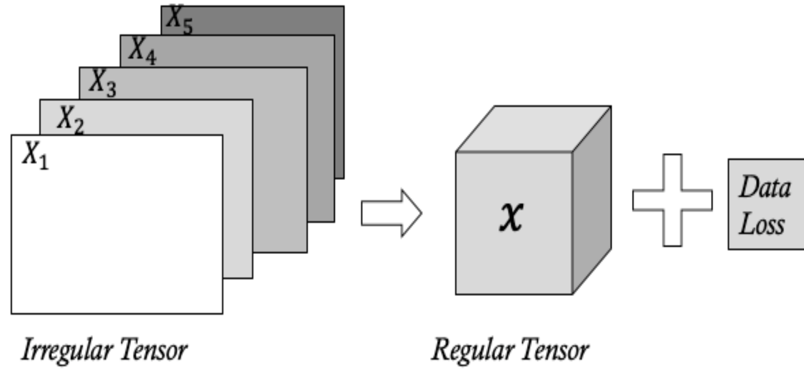

Figure 2. Irregular Tensor Factorization

2.4 Genetic Algorithm

Genetic algorithm is commonly used to generate high-quality solutions to optimize and search problems by relying on bio-inspired genetic operations such as mutation, crossover, and selection. The procedures of genetic algorithm are shown in *Figure 3*.

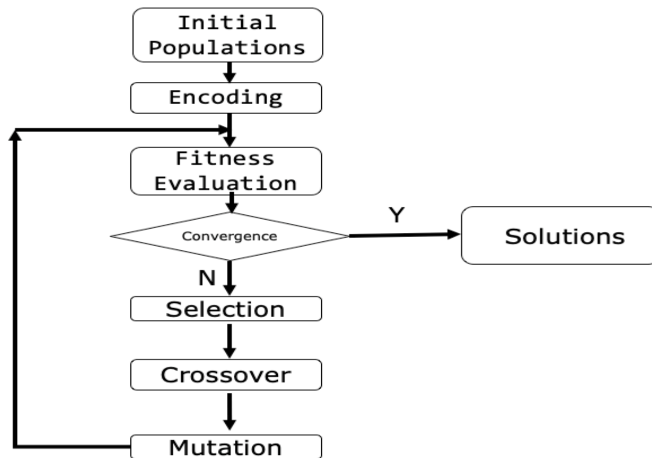

Figure 3. Procedures of Genetic Algorithms

In the step of initial populations, several sets of variables (parameters) are randomly selected within the search space. Each set of variables is called an individual or a chromosome, and each individual has randomness and fairness. In the step of encoding, individuals or

chromosomes are represented as a list in a certain way, and each element of the list is a gene. In the step of fitness evaluation, the objective function value of a question is used to measure the fitness of an individual. Once the fitness of all the individuals reach to a convergence, the individual with best fitness will be chosen as a solution. Otherwise, the fitness evolution and the operator of genetic algorithm will repeat until the fitness of all the individuals reach to a convergence.

The rest three procedures are the operators of genetic algorithm. The selection operator utilizes random rules to select individuals with high fitness value and to eliminate poor genes. Several pairs of individuals will be randomly selected as parents for gene exchange to generate children when processing the crossover operator of genetic algorithm. As for the mutation operator of genetic algorithm, each individual has a probability to be chosen, and each gene of the individual will change with a certain probability.

Those bio-inspired operators could be processed in parallel to speed up. To improve the algorithm, researchers have refined each bio-inspired operator. For example, roulette wheel selection [3] and tournament selection [8] have been proposed to improve the selection operator. Binary mutation and Gaussian mutation [9] have been proposed to improve the mutation operator. Even after it has been decades since the proposal of the genetic algorithm, it has been widely used up to these days.

2.5 Apache Spark

Although MapReduce has been implemented in large-scale data-intensive applications on commodity clusters successfully, some other applications do not suit to use MapReduce. Those applications contain many iterative algorithms which reuse a working set of data across multiple parallel operations. Therefore, a new framework named Spark has been distributed by Apache to support those applications. Spark contains two main new features to retain the scalability and fault tolerance of MapReduce. They are resilient distributed datasets [4] which store data on memory, and directed acyclic graph (DAG) of Spark [4] which achieves the fault-tolerance of Spark.

As shown in Figure 4.(a), Spark stores data in RDDs and those RDDs are divided into partitions that are processed on parallel. In addition, Spark RDDs have two kinds of operations for users, e.g., transformation and action operations. As shown the DAG of Spark in Figure 4.(b), it is a set of vertices and edges, where vertices represent the RDDs and the edges

represent the operations to be applied on RDD. When calling an action operation, the created DAG is submitted, and the results of the action operation are computed according to the DAG. When calling an actions operation, the created DAG is submitted and the results of the action operation are computed according to the DAG, therefore the fault-tolerance of Spark is achieved.

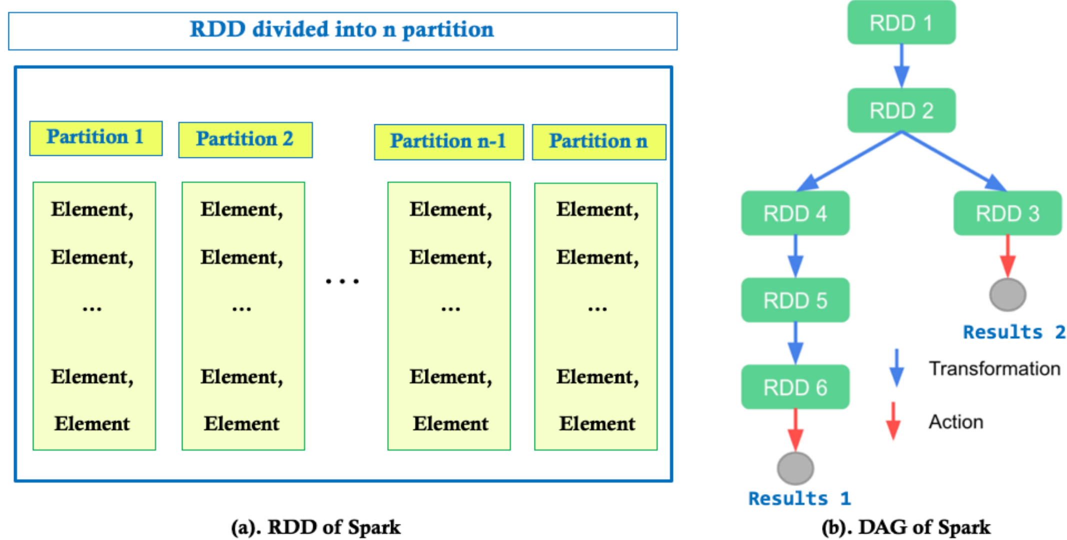

Figure 4. RDD and DAG of Spark

As shown in Figure 5, the architecture of a Spark cluster contains three components; they are Spark driver program, cluster manager and Spark workers. The Spark driver program converts the users' applications into tasks and schedule and send tasks on Spark workers. The cluster manager manages the resources of whole cluster as a plugin of Spark, and Spark can run different external cluster managers. A Spark worker contains a number of executors, which have multiple cores (≥ 1) inside, to execute tasks and return the results to the Spark driver program.

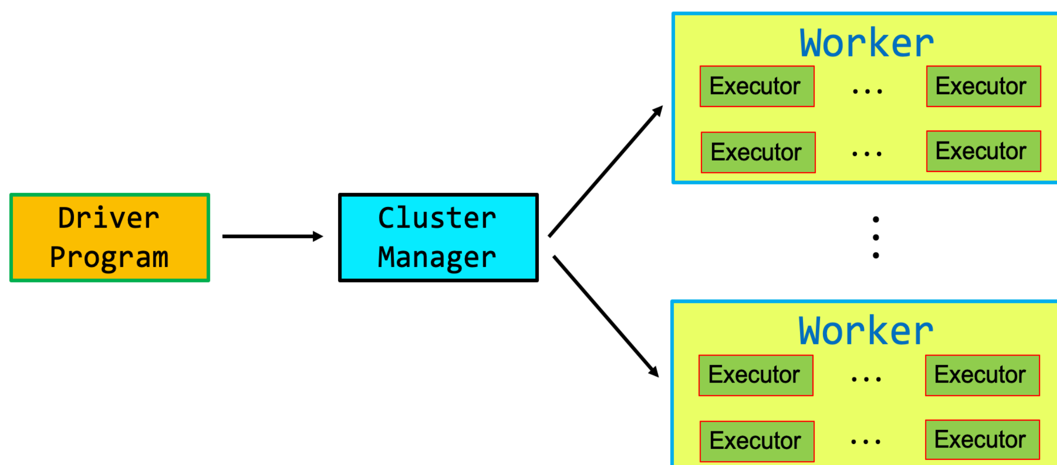

Figure 5. The Architecture of Spark cluster

An experiment conducted by the Apache official showed that Spark is 10x faster than Hadoop in iterative machine learning jobs. Recently, Spark has played an important role in the field of big data analytics ecosystem.

3. Related Work

3.1 Weighted Irregular Tensor Factorization (WITF)

Hu et al. proposed a weighted irregular tensor factorization (WITF) model [2] to leverage cross-domain feedback data across all users to learn the users' latent factors that are more accurate than the users' latent factors only in a particular domain. Epinions dataset [10], the dataset used to evaluate WITF, contains multiple domains while each domain contains different items. WITF considers the dataset as an irregular tensor, in which each domain contains different items, and then executes the tensor factorization to obtain the user's latent factors. According to the tensor theory, the irregular tensor must be transferred into a regular tensor, in which each domain contains the same items, to execute the tensor factorization.

Due to the inevitable data loss in irregular tensor transformation, WITF model utilizes a set of domain weights to reduce the data loss as shown in Figure 6.

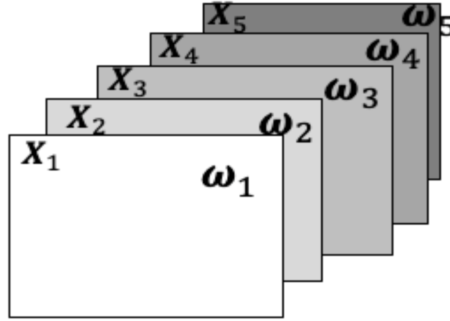

Figure 6. Domain Weights Configuration

The WITF model provides an objective function, which is based on CP decomposition, to describe the data loss. It is shown as Formula 1 [2], where ω_k is the domain weight for each domain X_k, and W_k is the weight for each entry in domain X_k. Σ_k is the diagonal matrix which is constructed based on the latent factor vector C_k which belongs to domain matrix C. P_k is the additional constraints [11] for obtaining the unique value of the Frobenius Norm ($\|\ldots\|_F$) in the objective function.

$$J = \underset{\boldsymbol{U},\boldsymbol{V},\boldsymbol{C}}{argmin} \frac{1}{2} \sum_{k=1}^{K} \omega_k \, \|\boldsymbol{W}_k \circledast (\boldsymbol{X}_k - \boldsymbol{U}\boldsymbol{\Sigma}_k(\boldsymbol{P}_k\boldsymbol{V})^T)\|_F^2 \qquad s.t. P_k^T P_k = \boldsymbol{I} \qquad (1)$$

WITF model adopts a specific set of domain weights $\{\omega_k\}$ and then processes iterations while updating U, V, C and P_k in each iteration until the objective function converges. The convergence value of the objective function could be considered as the minimal data loss of the irregular tensor factorization with a specific set of domain weights $\{\omega_k\}$. In addition, the WITF model also has proof that adopting the optimal domain weights configuration could minimalize the data loss.

Searching for the optimal domains weight configuration for the WITF model is a complex and time-consuming procedure. Therefore, WITF model simply adopts an empirical domain weights configuration which assigns the domain weights based on the ratios of ratings in each domain. Since the recommendations accuracy of WITF can be optimized by searching for the optimal domain weight configurations. A method to search the optimal domain weights configuration for WITF model is necessary.

4. Proposed Method

4.1 Overview

We propose a model which combines Spark and genetic algorithm to search for the optimal domain weight configurations for WITF model efficiently, then use the obtained optimal configuration to generate more precise user's latent factors to make recommendations. Here, Spark is an efficient and powerful large-scale data processing engine. Genetic algorithm is easy to be parallelized and widely used to search solutions for a complex problem, and it is suitable for complex computing problem such as searching optimal domain weights configuration for WITF model. In addition, WITF model is also suitable for parallelization. Therefore, the parallelization of genetic algorithm and WITF can speed up the computation when using genetic algorithm to search the optimal domain weights configurations for WITF model.

4.2 Architecture

Although genetic algorithm is easy to be parallelized and widely used to search the solutions for a complex problem, genetic algorithm is not suitable for problems with complex fitness evaluation. However, our proposed method utilizes the WITF model, which has a lot of complex computations, as the fitness evaluation. Therefore, we propose a two-phase Spark parallelization scheme to process WITF model and genetic algorithm efficiently.

The Spark RDDs are one of the core components of Spark. The RDDs store the data in memory to reduce the time of data I/O, and have various and efficient methods for users to use. Therefore, Spark users always separate the large data into partitions and store them into RDDs, then process those data on Spark in parallel.

For our proposed method, WITF and genetic algorithm operators, such as crossover and mutation, are the time-consuming parts. To speed up these operations, we process the WITF model as the first phase parallelization. In the first phase, the data of WITF, e.g., $\boldsymbol{X_k}$, $\boldsymbol{U}$, $\boldsymbol{V}$, and $\boldsymbol{C}$, are stored in RDDs and are computed in parallel inside the WITF model. This phase is shown as Algorithm 1

Algorithm 1: Parallelization of WITF Model [2]

$[\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{C}, \{\boldsymbol{P_k}\}]$ = **(function) Parallel_WITF**$(\{\boldsymbol{X_k}\}, \{\omega_k\})$, where $1 \leq k \leq \boldsymbol{K}$, $\boldsymbol{K}$ is the domain count

Input: $\{\omega_k\}$ is set of domain weights

$\boldsymbol{X_k}$ is the data matrix for each domain

Output: $\boldsymbol{U}$ is the latent factor matrix for users

$\boldsymbol{C}$ is the latent factor matrix for domains

$\boldsymbol{V}$, $\boldsymbol{P_k}$ are the latent factor matrices for items

---

Begin

Initialization:

1: Randomly Initialize $\boldsymbol{U}, \boldsymbol{C}$

2: $\boldsymbol{V} = \boldsymbol{I}$

3: $\boldsymbol{P_k} = \boldsymbol{AB^T}$, with the SVD: $\boldsymbol{X_k^T U \Sigma_k V^T} \approx \boldsymbol{A \Sigma B^T}$

Iteration:

4: Generate tensor with each $\boldsymbol{X_k}$

5: Update $\boldsymbol{U_i}$ of each user $\boldsymbol{i}$ in parallel by WITF theory

6: Update $\boldsymbol{C_k}$ of each domain $\boldsymbol{k}$ in parallel by WITF theory

7: Update $\boldsymbol{V}$ as a whole by WITF theory

8: Update $\boldsymbol{P_k}$ of each domain $\boldsymbol{k}$ in parallel by WITF theory

Repeat 4-8 until the objective function convergence

9: Return $\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{C}, \{\boldsymbol{P_k}\}$

End

The second phase parallelization is for genetic algorithm operators. We use the RMSE (root mean square error) [12] metrics, which is calculated by using the returned $\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{C}, \{\boldsymbol{P_k}\}$ from WITF model, as a fitness value of each individual ($\{\omega_k\}$). The RMSE is defined as Formula 2, where Y_i is the user's real rating, i.e, ground truth, for item $\boldsymbol{i}$, and $\widehat{Y_i}$ is the predicted rating of the user to item $\boldsymbol{i}$. In general, the smaller value of the RMSE represents better accuracy of the recommendations.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N}(Y_i - \widehat{Y_\iota})^2}{N}} \qquad (2)$$

We store the individuals in RDDs and process those operators in parallel on Spark. This phase is shown as Algorithm 2.

Algorithm 2: Parallel Genetic Algorithm with WITF

Input: $\{\omega_k\}$ is the initial set of domain weights

$\boldsymbol{X_k}$ is the data matrix for each domain

$\boldsymbol{G_{max}}$ is the maximum number of generations of genetic algorithm

Output: $\{\boldsymbol{\psi_k}\}$ is the optimal set of domain weights

---

Begin

***Initial Population*:**

1: Make initial population by $\{\omega_k\}$

***Generation Iteration of Genetic Algorithm*:**

2: **for each** individual $\{\omega_k\}$ in population **do**

3: **[$\boldsymbol{U}$, $\boldsymbol{C}$, $\boldsymbol{V}$, $\boldsymbol{P_k}$] = Parallel_WITF**($\{\boldsymbol{X_k}\}$, $\{\omega_k\}$)

4: Calculate RMSE by **[$\boldsymbol{U}$, $\boldsymbol{C}$, $\boldsymbol{V}$, $\boldsymbol{P_k}$]**

5: Save the RMSE as the fitness value of $\{\omega_k\}$

6: **end for**

7: Selection of the population in parallel

8: Crossover of the population in parallel

9: Mutation of the population in parallel

***Repeat 2-9 for $\boldsymbol{G_{max}}$ times*:**

10: $\{\boldsymbol{\psi_k}\}$ is the $\{\omega_k\}$ which obtain minimal RMSE

11: Return $\{\boldsymbol{\psi_k}\}$

End

---

5. Experimental Evaluation

5.1 Datasets and Environment

The Epinions datasets [10] is extracted from the Epinions website, which is an e-commerce website and has multiple domains of products. The users can review ratings, which are integers from 1 to 5, to products of different domains, and products on each domain are unique to each domain. Therefore, we select five domains which have most ratings from the Epinions datasets and pre-process the data to discard users who are lack of feedbacks on those five selected domains. Table 2 shows the two sizes datasets we used in our experiment; the Table 2.(a) shows the statistic of the "small" dataset and Table 2.(b) shows the statistic of the "large" dataset. In the "small" dataset, we first remained the users who have at least ten ratings in each selected domain, and then remained the items which have been rated by at least ten remained users. Similar with the "small" dataset, we remained the users who have at least five ratings in each selected domain, and then remained the items which have been rated by at least five remained users in the "large" dataset.

Table 2. Two sizes datasets

(a). "small" Dataset Statistic

	#Users	#Items	#Ratings
Domain 1	2,403	1,104	14,960
Domain 2	2,403	320	7,350
Domain 3	2,403	1,362	30,490
Domain 4	2,403	1,050	10,223
Domain 5	2,403	761	8,733

(b). "large" Dataset Statistic

	#Users	#Items	#Ratings
Domain 1	6,682	1,771	27,786
Domain 2	6,682	702	13,802
Domain 3	6,682	2,318	47,972
Domain 4	6,682	2,459	22,448
Domain 5	6,682	1,786	18,925

We process the experiment on four servers, and the environment of each server is shown as Table 3. In our experimental Spark cluster, one server has been configured as the Spark master server to process the Spark driver program and Spark cluster manager. The other three servers have been configured as the Spark workers, which are also named as the Spark slaves servers, to process Spark tasks.

Table 3. Experiment Environment of each server

OS	CPU	#CPU(Core)	Memory
CentOS 7.5	Intel Xeon CPUE5-2620 v4 @ 2.10GHz	2 Physical 16 Logical	128GB

5.2 Implementation of Genetic Algorithm

5.2.1 Initial Population and Encoding

The genetic algorithm has two major encoding methods, binary-encoding [3] and float-encoding [13]. The binary-encoding method represents an individual as a series of binary values. On the contrary, the float-encoding method, which is also known as real-value encoding, represents an individual as a series of float values. Compare with the binary-encoding method, the float-encoding method is suitable for questions that have a large search space which is similar with searching the optimal domain weights for the WITF model. Therefore, we adopted the float-encoding method for our implementation to make the initial population. It is shown in Figure 7.

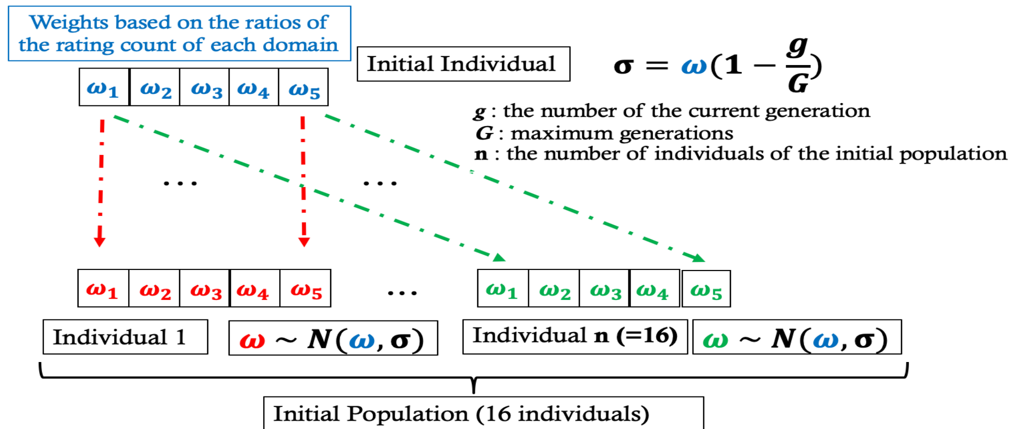

Figure 7. Initial Population

The individuals (chromosomes) of the initial population, where the individual count $\mathbf{n}$ was set as 16, in genetic algorithm is a set of domain weights $\{\omega_k\}$. Due to the dataset we used has five domains, each individual has five domain weights, where each domain weight $\boldsymbol{\omega_k}$, $1 \leq k \leq \mathbf{5}$, is considered as a gene inside an individual (chromosome).

The WITF model can obtain a specific RMSE value with the domain weights based on the ratios of rating counts of each domain, and the domain weights configuration is represented as the initial individual with the "blue" genes in the Figure 7. In addition, the search space of the domain weights is very large. Therefore, to reduce the search space, we utilized the "blue" individual as the initial individual, and each individual, such as "red" individual, "green" individual in Figure 7 and so on, of the initial population was mutated from the "blue" individual. The mutation algorithm is the Gaussian Mutation [9] which was also adopted as the algorithm of the mutation operator of genetic algorithm for our proposed method.

Based on the mutation algorithm, as shown in Figure 7, each gene ω_k in an individual is mutated from the gene ω_k in the "blue" individual based on the normal distribution $\boldsymbol{N(\omega, \sigma)}$, where $\boldsymbol{\omega}$ is gene ω_k in the "blue" individual. Since the current generation is the initial, the $\boldsymbol{g}$ in the Figure 7 is 0, and the $\boldsymbol{\sigma}$ turns to be $\boldsymbol{\omega}$.

5.2.2 Fitness Valuation

As our proposed method was proposed for searching the optimal domain weights configurations for the WITF model, the fitness value of each individual becomes as the recommendation accuracy (RMSE) of the WITF model by using the domain weights in the individual.

5.2.3 Selection Operator

We adopted the tournament selection [8], which is shown as Figure 8, as the selection operator of genetic algorithms. The tournament selection first selects $\boldsymbol{K}$ individuals at random and then selects the individual, which has the best fitness among the $\boldsymbol{K}$ selected individuals, as a parent of the next generation. After selected a parent, tournament selection put back all the $\boldsymbol{K}$ selected individuals and repeats those two step until it has selected enough parents for the next generation.

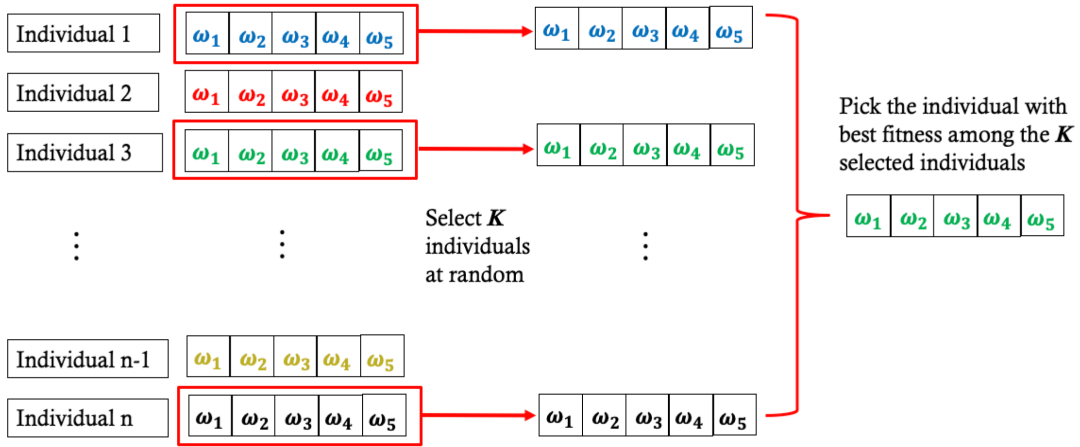

Figure 8. Tournament Selection

5.2.4 Crossover Operator

We adopted the whole arithmetic recombination [14], which is shown in Figure 9, as the crossover operation of genetic algorithm. The whole arithmetic recombination is the most commonly used crossover operator for float-encoding genetic algorithm and works by taking the weighted sum of the two parental alleles for each gene in children. The parameter α is in the interval $(\mathbf{0}, \mathbf{1})$, and we let the $\alpha = 0.25$ which is usually used.

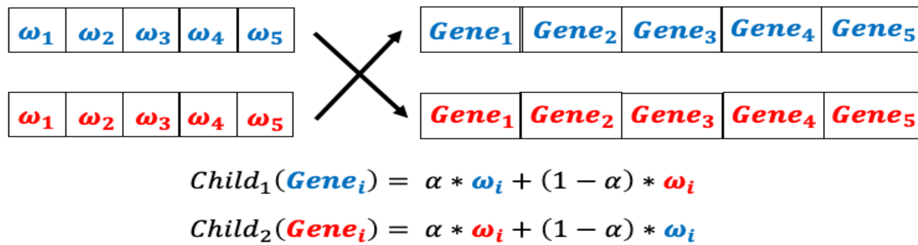

Figure 9. Crossover based on Whole Arithmetic Recombination

5.2.5 Mutation Operator

As we have discussed in section 5.2.1, the Gaussian mutation [9] is used as the mutation operator which is shown in Figure 10. Each individual has a probability for processing mutation and we adopted the probability as 0.5. In addition, an individual, which will be processed mutation, also has a probability of each gene for processing mutation and we adopted the probability as 0.2 in our experiment. Due to the large of the search space, the used values of two probabilities are larger than usually used values for obtaining new individuals easily.

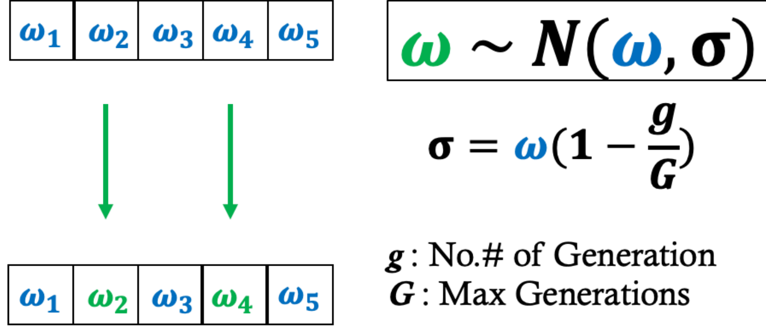

Figure 10. Mutation based on Gaussian Mutation

5.3 Parallelization Evaluation of WITF model

5.3.1 Parallelization Granularity of WITF model

In a Spark cluster, many computing resources are contained and considered as executors. To manage those executors in the cluster, Spark provides a driver program to send tasks, which contain both data and instructions, and receives the results when executors finish their assigned tasks. An efficient program on Spark should execute its procedures on the Spark executors as long as possible.

Among the procedures of our proposed method, the most time-consuming procedures are the fitness evaluation, which compute the RMSE value of the WITF model by using the domain weights configuration of each individual. As we have discussed the WITF model in section 3.1, WITF model executes iterations to update its parameters, e.g., $\boldsymbol{U}$, $\boldsymbol{C}$, $\boldsymbol{V}$, and $\boldsymbol{P_k}$, until reaching the convergence. Therefore, the efficiency of each iteration of WITF model is important for the overall efficiency of our proposed method.

The "small" dataset we used has five selected domains and 2,403 users. Based on the WITF model, the procedures to update parameters $\boldsymbol{C}$ and $\boldsymbol{V}$, which are described as the step 6 and step 7 in Algorithm 1 of section 4.2, have to process a two-level loop. The first-level loop is the loop for the five selected domains, and each domain has a second-level for the 2,403 users. We first utilized the users as the element of Spark RDDs. Due to the two-level loop of the procedures to update $\boldsymbol{C}$ and $\boldsymbol{V}$, we implement five RDDs for each procedure. As it is shown in Figure 11, each RDD of domain stores all 2,403 users as elements and divides users into a number of partitions. When updating the $\boldsymbol{C}$ and $\boldsymbol{V}$ on each domain, the Spark driver program

sends those partitions and broadcast necessary data to Spark executors.

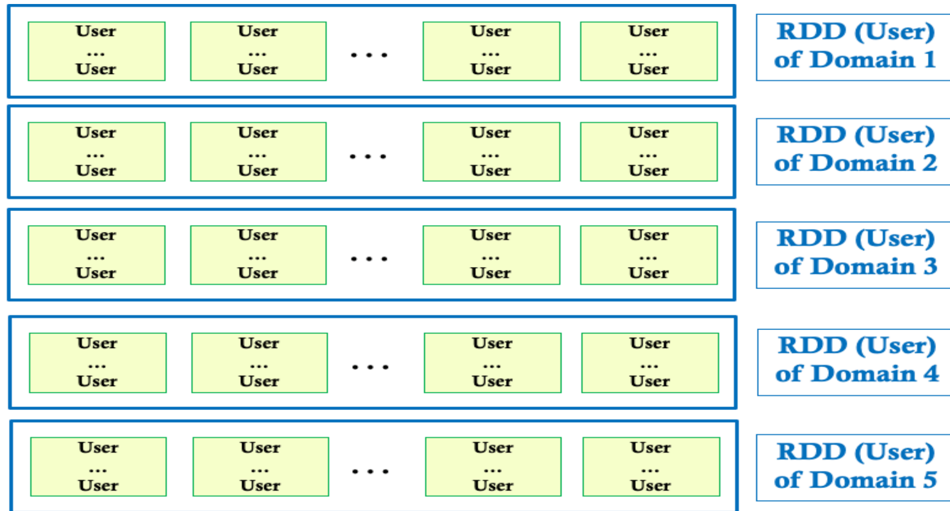

Figure 11. RDDs for users in each domain

As we implement the WITF model on Spark by using Spark Python API and web UI provided by Spark to monitor the program run on Spark, we can analyze the timeline of an iteration of WITF model we implement by using the "small" dataset on a server with 16 executors which have one core in each executor. The timeline is shown as Figure 12:

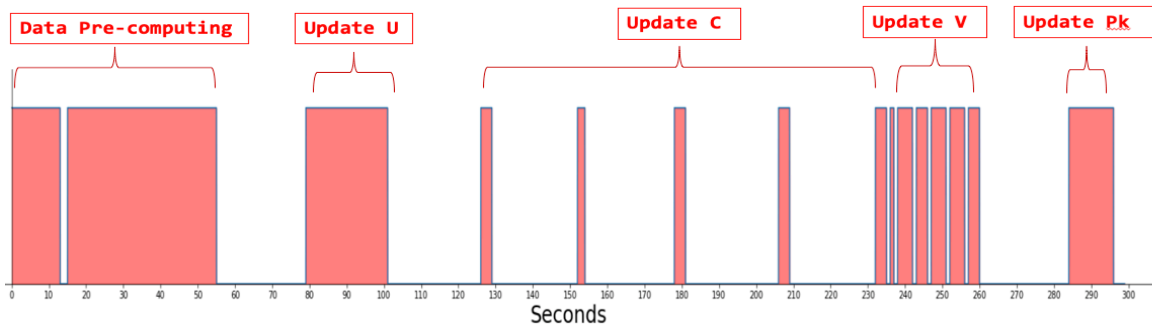

Figure 12. Timeline of a WITF iteration by using domain RDDs on Spark Cluster ("small" datasets, 16 executors, 1 core in each executor)

The "red" bars in Figure 12 represent the period of Spark executors executing the tasks. On the contrary, the rest parts in Figure 12 represent the period when Spark driver program is executing and the Spark executors are idle. The total execution time of a WITF iteration by using domain RDDs is 296 seconds, and time ratio between executors executing time with a WITF iteration executing time is just 48.8%. According to the timeline, when processing the procedure of update $\boldsymbol{C}$ on each domain, the Spark driver program spent too much time to send RDDs partitions and broadcasted necessary data to the Spark executors.

To reduce the execution time of Spark driver program, we have adopted a different way to construct the RDDs to update parameters $\boldsymbol{C}$ and $\boldsymbol{V}$. As it is shown in Figure 13, we re-

constructed the two-level loop as a one-level loop, so that each element in the one-level loop consists of a pair of domain and user. Those pairs are also stored as the element of a whole pair RDD [4].

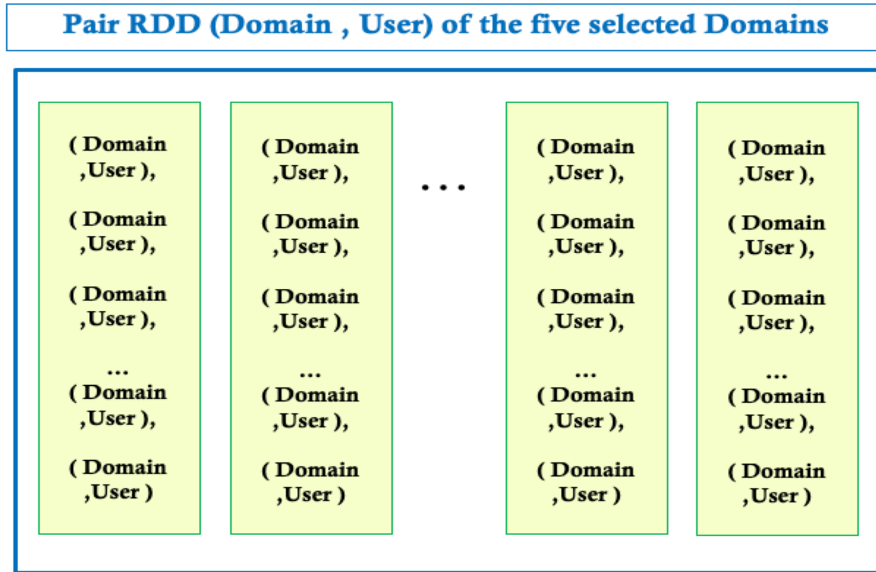

Figure 13. Pair RDD

The timeline of a WITF iteration by using the Pair RDD on Spark is shown as Figure 13. It is the same as Figure 11, the gray bars in Figure 14 represent the period of Spark executors executing the tasks. On the contrary, rest parts in the figure represent the period when Spark driver program is executing and executors are idle. The total execution time of one time WITF iteration by using the pair RDDs is 224 seconds reduced from 296 seconds. The ratio of time between executors executing time with an WITF iteration executing time is 67.9% improved from 48.8%. The improvement shows that choose a suitable parallelization granularity is important for an efficient parallel computing application.

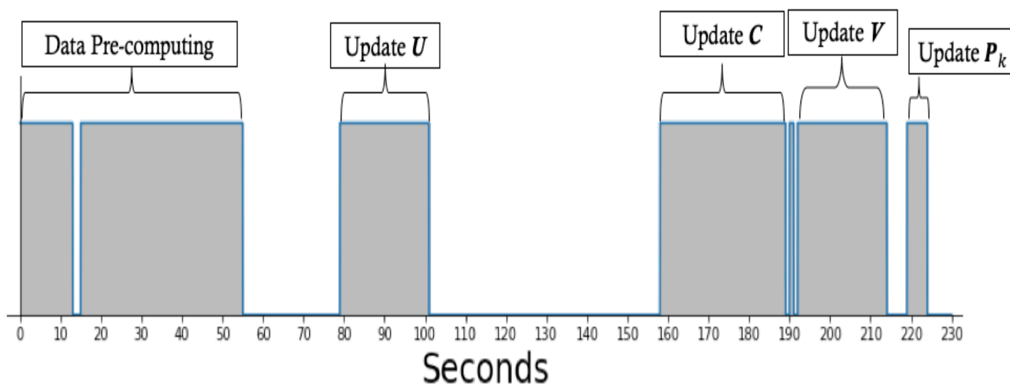

Figure 14. Timeline of a WITF iteration by using pair RDDs on Spark Cluster (“small” datasets, 16 executors, 1 core in each executor)

Due to the Spark driver program having to broadcast large-scale data to the executors, the execution time of Spark driver program is still too long between the procedure "Update ***U***" and procedure "Update ***C***".

5.3.2 Efficiency Evaluation of WITF model on Spark Cluster

We have processed a series of experiments by using the two sizes datasets to evaluated our implementation of the WITF model on our Spark cluster, which contains one master server and three slaves servers (16 cores in each server, 48 cores in total). We first processed experiments with different numbers of Spark executors which contain one core inside. Then, we processed experiments with different numbers of Spark executors which contain multiple cores inside and all 48 cores are used.

5.3.2.1 Executors with one core inside

First, we processed three group experiments for the two sizes of datasets by setting 16, 32 and 48 executors with one core inside. The results of execution time (minutes) of a WITF iteration is shown in Figure 15. According to the results, the execution time of a WITF Iteration has a slight increase with the increase of the used Spark executors while using either "small" or "large" dataset.

Figure 15. Execution Time of a WITF Iteration

The reason, why the execution time was increase when we utilized more executors (cores), is that those executors only contain one core inside and the computation ability of each executor

was limited. In addition, the communications between the Spark driver program and the Spark executors, e.g., data broadcast and task results return, were increased by utilizing more executors. Therefore, the execution time of a WITF iteration was increased even we have utilized more computation resources.

5.3.2.2 Executors with multiple cores inside

Since the experiment results by setting the Spark executors with one core inside show that the executor computation ability is limited, we have processed a series of experiments of the two sizes of datasets by setting the Spark executors with multiple cores inside (48 cores in total). The results of the execution time of a WITF iteration for the two datasets are shown in Figure 16.

According to the results in Figure 16, the execution time of a WITF iteration is decreased with the increase of cores in each executor and the with the decrease of executors. The execution time tends to be stable when the core's number is more than eight in each executor (the executor's number is less than six, 48 cores in total). When using the "large" dataset, the best execution time, which was obtained with three executors (16 cores in each), is 59.96% faster than the worst execution time which was obtained with 48 executors (one core in each). When using the "small" dataset, the best execution time, which was obtained with three executors (16 cores in each), is 40.04% faster than the worst execution time which was obtained with 48 executors (one core in each).

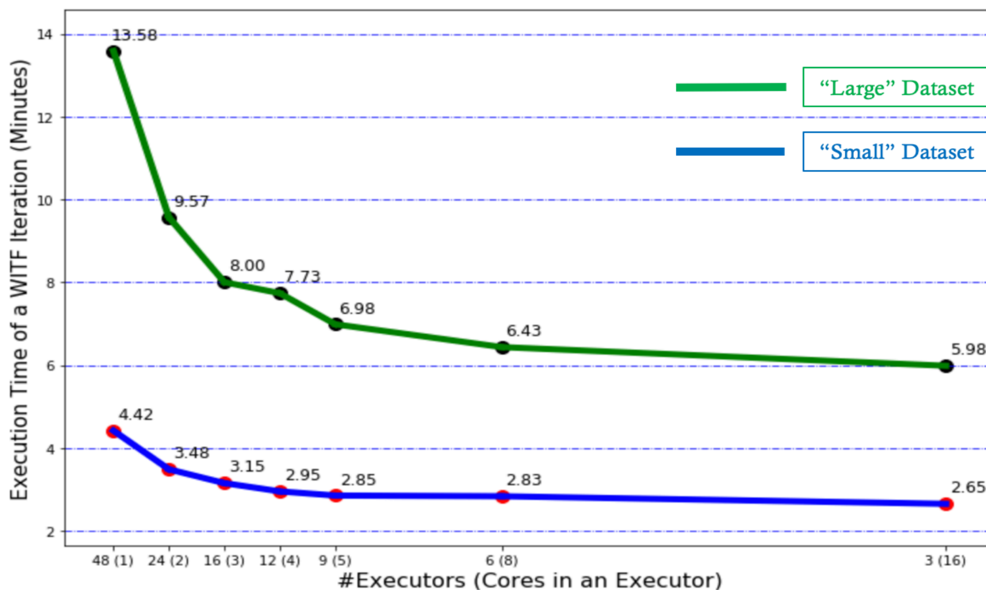

Figure 16. Execution Time of a WITF Iteration with different multiple cores Spark executors

As for the ratio between the executing time on executors with the whole execution time of a WITF iteration, the results for the two sizes of datasets are shown in Figure 17. Similar with the result in Figure 16, the execution time ratios are also decreased with the core's numbers increase in each executor and the executor's numbers decrease, and also tends to be stable when the core's number is more than eight in each executor (the executor's number is less than six, 48 cores in total).

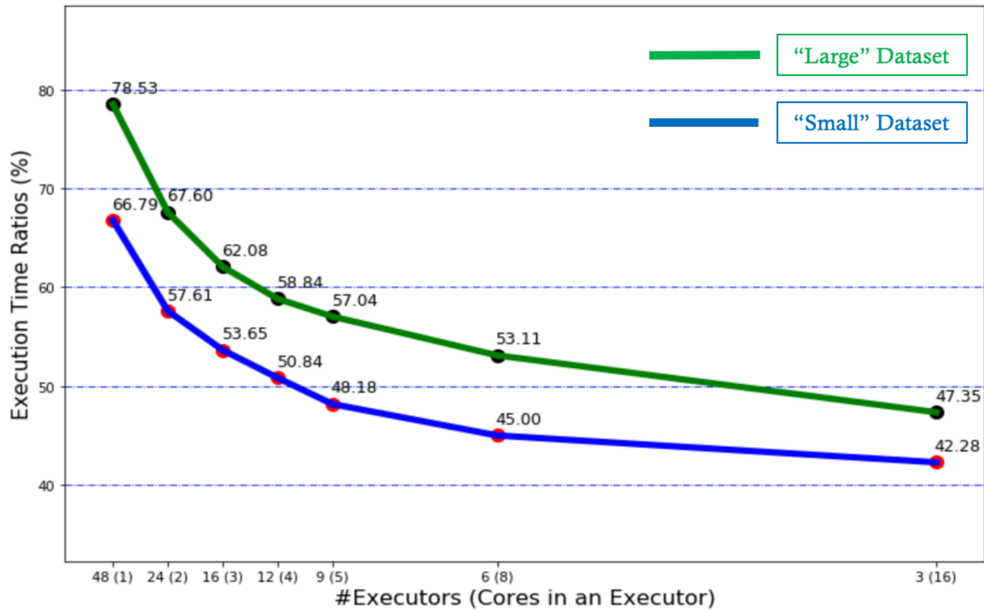

Figure 17. Execution Time Ratios of a WITF Iteration with different multiple cores Spark executors

The results in Figure 16 and 17 show that the more cores in each Spark executor could increase the computation ability of each Spark executor and reduce the execution time of the tasks processed on each Spark executor. Even the less executors could reduce the communications between the Spark driver program and then reduce the execution time on the Spark driver program, the execution time ratio, which is between the time on the Spark executors with the total execution time, is still decreased. It means that the decrease of the Spark driver program execution time is much less than the decrease of the Spark executors execution time while setting more cores in each executor and less executors in our experiments. In other words, our implementation of the WITF model could have better efficiency performance in the Spark cluster with high computation ability executors.

5.4 Evaluation of Execution Time

We initialized the population of genetic algorithm with 16 individuals and set three

executors (16 cores in each, 48 in total), and then execute our implementation of proposed method for 50 generations by using the two sizes of datasets. Based on the execution time, we obtain the execution time statistics shown as Table 4:

Table 4. Genetic Algorithm Execution Time

(a). "Small" Dataset

Procedure	Average Execution time (min.)
1 Generation	172.62
1 Individual	10.63

(b). "Large" Dataset

Procedure	Average Execution time (min.)
1 Generation	382.93
1 Individual	23.93

5.5 Genetic Algorithm Generation Evaluation

We also have calculated the mean fitness (RMSE) value of all 16 individuals in each generation, and record the minimum fitness (RMSE) value as the best RMSE of all 16 individuals in each generation. In addition, we chose the RMSE value calculated by using the WITF model's empirical domain weights configuration, which decides the domain weights by the number of ratings in each domain, as the baseline.

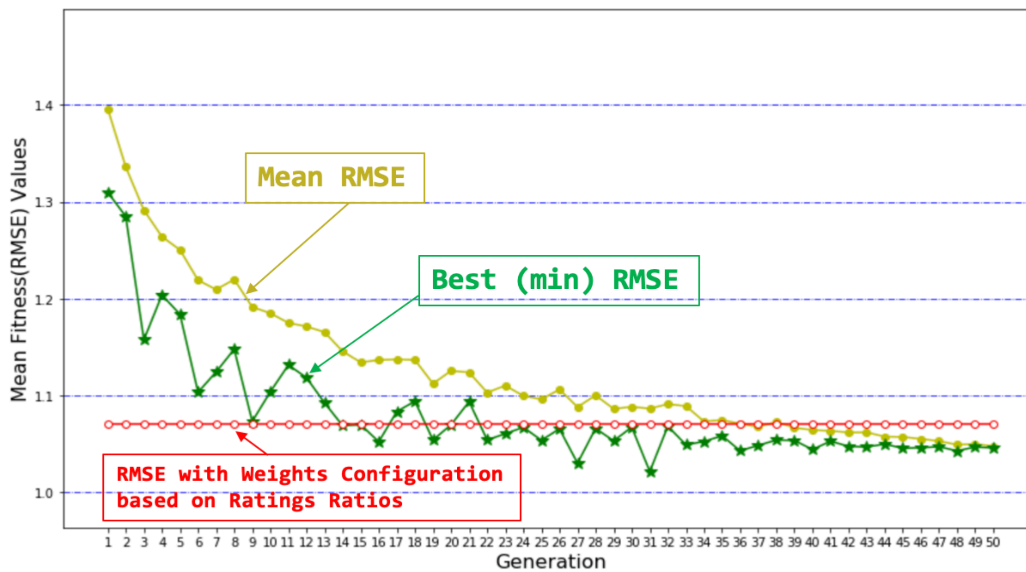

Figure 18. Genetic Algorithm Generations Evaluation by using the "small" dataset

For the “small” dataset, the comparison of those three value in each generation is shown in Figure 18. The “red” line represents the baseline. The “golden” polyline represent the mean fitness (RMSE) value of each generation, and the “green” polyline represent the best fitness (RMSE) value of each generation. As we can see on the figure, the mean and best fitness (RMSE) value of each generation tend to be stable and better than initial individuals mean and best fitness (RMSE) values with the generation grows. After the 22nd generation, the best fitness (RMSE) values are better than the baseline, and tend to be stable after 41st generation. Before the 34th generation, the mean fitness (RMSE) values are worse than the baseline, then after the 39th generation, the mean fitness (RMSE) values are better than the baseline and approach to the best fitness (RMSE). That represents that the search result of genetic algorithm reaches to convergence in our proposed method. In addition, the best RMSE by using the “small” dataset among all generations has been improved by 4.2% comparing with the baseline.

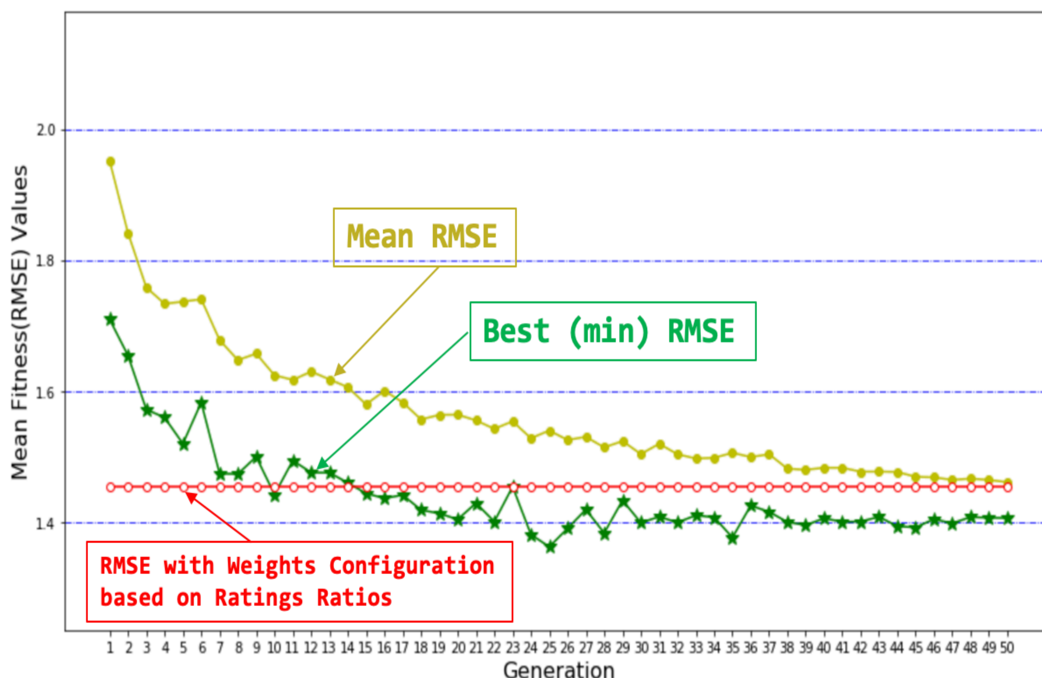

Figure 19. Genetic Algorithm Generations Evaluation by using the “large” dataset

For the “large” dataset, the result, which is shown in Figure 19, is similar with the results of the “small” dataset. Due to the “large” dataset is sparser than the “small” dataset, the baseline, mean fitness (RMSE) and best fitness (RMSE) are worse than the fitness value for the “small” dataset, and the mean fitness (RMSE) are worse than the baseline among all the 50 generations. However, as similar with the result of the “small” dataset, the mean fitness (RMSE) would approach to the best fitness (RMSE) after enough generations. After the 14th generation, the best fitness (RMSE) values are better than the baseline, and then tend to be stable after 38th

generation. In addition, the best RMSE by using the “large” dataset among all generations has been improved by 6.3% comparing with the baseline.

The result shows that our proposed method has the ability to search and obtain the optimal domain weights configuration for WITF to make recommendations which has the best RMSE.

6. Conclusion

In this paper, we proposed a method which combines the WITF model and genetic algorithm to search the optimal domain weights configuration of WITF model for more accurate recommendations. To make the computation efficiently, we adopted parallelization into both the WITF model and genetic algorithm, then implemented and evaluated our proposed method on Spark platform.

We evaluated two sizes of datasets on a Spark cluster which contains 48 cores, and analyzed the experiment results of each dataset. The evaluation results showed that our method and implementation has the ability to search the optimal domain weights configuration for WITF model. For the "small" dataset, our proposed method has searched a domain weight configuration with 4.2% RMSE improvement. For the "large" dataset, our proposed method has searched a domain weight configuration with the 6.3% RMSE improvement.

The efficiency evaluation results show that parallelization implementation of the WITF model could have better efficiency performance in the Spark cluster with high computation ability executors. However, this method should be compared the accuracy of recommendations with several baseline methods in the future.

References

[1] P. Resnick, N. Iacovou, M.Suchak, P.Bergstrom, and J.Riedl, GroupLens: An Open Architecture for Collaborative Filtering of Netnews, Proceedings of ACM Conference on Computer Supported Cooperative Work (CSCW1994), pp.175-186, 1994.

[2] L. Hu, L. Cao, J. Cao, Z. Gu, G. Xu, and D. Yang. Learning informative priors from heterogeneous domains to improve recommendation in cold-start user domains, ACM Trans. Inf. Syst. Vol.35, No.2, Article 13, 2016.

[3] D.E. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley. 1989

[4] Zaharia M, Chowdhury M, Das T, Dave A, Ma j, McCauley M, Franklin M, Shenker S, Stoica I, Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing, Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation, Berkeley, 2012.

[5] B. Li. Cross-domain collaborative filtering: A brief survey, Proceedings of the 2011 IEEE 23rd international Conference on Tools with Artificial Intelligence, 2011.

[6] W. Pan, E. W. Xiang, N. N. Liu, and Q. Yang, Transfer learning in collaborative filtering for sparsity reduction. Proceedings of the 24th AAAI Conference on Artificial Intelligence, 2010.

[7] T. G. Kolda and B. W. Bader, Tensor decompositions and applications, SIAM Rev., Vol.51, No.3, pp.455–500, 2009.

[8] D. E. Goldberg and K. Deb, A comparative analysis of selection schemes used in genetic algorithms, Foundations of Genetic Algorithms, Vol.1, pp.69-93, 1991.

[9] T. Back and H.-P. Schwefel, An Overview of Evolutionary Algorithms for Parameter Optimization, Evolutionary Computation,Vol.1, No.1, pp.1-23, 1993.

[10] S. Meyffret , E. Guillot , L. Médini, and F. Laforest, RED: a Rich Epinions Dataset for Recommender Systems, Dataset for Recommender Systems, 2014.

[11] H. A. L. Kiers, J. M. F. Ten Berge, and R. Bro. Parafac2—part I. A direct fitting algorithm for the parafac2 model, Journal of Chemometrics, Vol.13, pp.275-294, 1999.

[12] CJ. Willmott, K. Matsuura, Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance, Climate Research, Vol.30,

No.1, pp.79-82, 2005.
[13] C. T. Su and W. T. Tyen, A Genetic Algorithm Approach Employing Systems, Proceedings of International Congress on Modeling and Simulation, pp. 1444-1449, 1997.
[14] Z. Michalewicz. Genetic Algorithms + Data Structures = Evolution Programs. Springer, 3rd edition, 1996.
[15] B. Sarwar, G. Karypis, J. Konstan, J. Riedl, "Item-Based Collaborative Filtering Recommendation Algorithms", Proceeding of the 10th international conference on World Wide Web, WWW 2001. 2001

Acknowledgement

I would like to give my deepest gratitude to Professor Yamana, who has given lots of comments and instructions for my research and my master thesis. Besides the comments and instructions about research, Professor Yamana also have make many chances for me to attend and join the projects which could improve my ability.

Also, I would like to give my thanks to Satoshi Hasegawa and Takumi Zamami, who are the member of DMM.com company, for receiving many advice about my research.

Last but not least, I would like to thank all the lab members. I have got many advice from them on each seminar and group discussion. It is my great honor to be one of the Yamana Lab.

Publication

Guodong Xue, Seiki Miyamoto, Takumi Zamami, Hayato Yamana. “A Lossless Irregular Tensor Factorization for Cross-domain Recommender System with Genetic Algorithm on Spark” in DEIM 2019