Efficient implementation of movie recommendation system by using Genetic Algorithm

Kai-Yun Ku
Department of Computer Science,
National Chiao Tung University
University Road NCTU Hsinchu City
Kevin706130424@gmail.com

Yu-Jin Yang
Department of Computer Science,
National Chiao Tung University
University Road NCTU Hsinchu City
ch23173885@yahoo.com.tw

ABSTRACT

We implement a recommendation system according to the paper [2]. Our recommendation system which uses genetic algorithm to find importance of different personal preferences of users, then the system can find the similar neighborhood of current user and recommend movie which the user in neighborhood like to current user. We use OpenMP to parallel our recommendation system.

KEYWORDS

Genetic algorithm, recommendation system, parallel computing

1 INTRODUCTION

Recommendation system is to recommend or suggest items to the customer based on his/her preferences. Our work focus on the use of evolutionary way to construct a movie recommender system. The main concept is that: if we can find other users who are similar to the current user (In this paper, we refer to the current user as the active user, A) by comparing their preferences, then the movie those users like you may also like. So our system will recommend some high rating movies from those similar neighborhoods to the current user who uses recommendation system. The reason we use genetic algorithm is that we don't need to decide how important each feature is. The genetic algorithm will find importance of different features and give the weight to them. So we can use the set of weights to find good similar neighborhood and finish the recommendation. Finally, we use omp4j (OpenMP for Java) to parallel our program and compare the different experiment result.

2 PROPOSED SOLUTION

2.1 System overview

Our system is based around a collaborative filtering approach, building up profiles of users and movies then using an algorithm to find profiles similar to the current user. In this work, the MovieLens dataset (http://grouplens.org/datasets/movielens/), was used for our experiments. We chose MovieLens 100K Dataset, this dataset contains 100000 ratings by 943 users on 1682 movies. The data was collected through the MovieLens web site during the seven-month period from September 19th, 1997 through April 22nd, 1998. Each user had rated at least 20 movies. Our evolutionary recommender system uses 22 features including user and movie features from this data set as following: movie rating, age, gender, occupation and 18 movie genre frequencies: action, adventure, animation, children, comedy, crime, documentary, drama, fantasy, film-noir, horror, musical, mystery, romance, sci-fi, thriller, war, western. We partition all movie items from MovieLens dataset into two datasets, a training set (1/3) and a testing set (2/3).

2.2 Training Stage

In training stage, each individual in each iteration runs following flow chart Fig. 1.

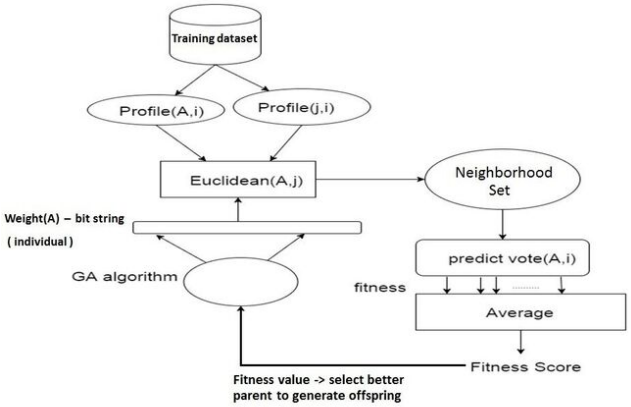


Figure 1: Training stage flow chart

The training stage system architecture consists of several main parts and we are going to introducing all of them:

2.2.1 Profile. We can get those feature from database we mention above to generate each user profile. We define profile(j,i) to mean the profile for user j on movie item i (see Fig. 2). Every profile(j,i) contains not only the movie i feature but user j personal information. The profile(j) is therefore a collection of profile(j,i) for all the items i that j has seen. Note that the genre frequencies stand for that all

movie types the user j had seen. If user j had seen action movie then the corresponding genre bit value is 1, otherwise is 0. Once profiles are built, given an active user A, a set or neighborhood of profiles similar to profile(A) can be found.

2.2.2 Genetic Algorithm. We use genetic algorithm to select good set of weights which convert into bit string as an individual. We initially create 100 random individuals (population size = 100) as first generation. When creating a new generation, we randomly selected the individuals out of the top 40% of the whole population to be parents. Two offspring are produced from every pair of parents, using uniform crossover with uniform rate 0.5. Mutation is applied to each locus in genotype with probability 0.01. And the fitness value of each individual will be introduced in the Fitness Value subsection.

2.2.3 Euclidean. For the active user A and other users in the system, we compute the distance or dissimilarity between the selected profiles from all other user and the active user's profile using a distance function. We first define $diff_{i,f}(A,j)$ which has five elements and each element store value according to comparing same feature to two distinct user profiles (see Fig. 3). We first compute $diff_{i,f}(A,j)$ for all user j except active user A. If the active user A and user j have the same feature then corresponding element in $diff_{i,f}(A,j)$ has value 0, otherwise 1. Especially the last feature genre frequencies in profile, we compute how many same genres those two profiles have. And we store the value which is same genres number divide total genres number into last element $diff_{i,f}(A,j)$.

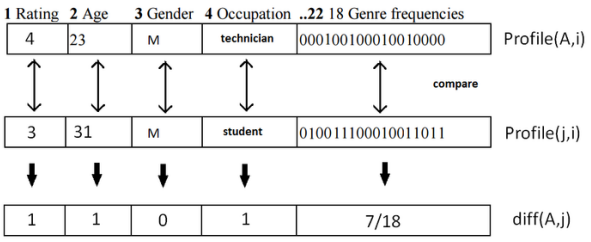


Figure 3: $diff_{i,f}(A, j)$

Every individual is represented as a set of weights (see Fig. 4), where wf is the weight associated with feature f whose genotype is a string of binary values. We use 8 bits string to represent a single weight. We can convert from bit string into corresponding decimal value (weight value).

Figure 4: Phenotype of an individual in the population

The diff can now be conducted using a modified Euclidean distance function. Euclidean(A,j) is the dissimilarity between active user A and user j :

$$euclidean(A,j) = \sqrt{\sum_{i=1}^{z}\sum_{f=1}^{22} w_f * diff_{i,f}(A,j)^2}$$

where:

A : the active user

j : a user, where $j \neq A$

i : a common movie item, where profile(A,i) and profile(j,i) exist

wf : the active user's weight for feature f

$diff_{i,f}(A,j)$: the difference in profile value for feature f between user A and j on movie item i

2.2.4 Neighborhood set. For each individual, we can compute Euclidean(A,j) for all user $j \neq A$. And we get the top 20 Euclidean value of all Euclidean(A,j) as a neighborhood set of active user A. All user in neighborhood set is similar to active user A according to those weights which is represented by each individual. Note that if we find good weights, users in neighborhood set are actually similar to active user A. So we use genetic algorithm to find good weights.

2.2.5 Fitness value. As we mentioned previously, a poor set of weights might result in a poor neighborhood set of profile for the active user and hence poor recommendations. A good set of weights should result in a good neighborhood set, and hence good recommendations. So how to judge the quality of the neighborhood set and recommendations is required, in order that a fitness score can be assigned to the corresponding weights. Given the active user A and a set of neighboring profiles, it is possible to predict what A might think of them. That is a certain movie is suggested because similar users saw it, but those users only thought the movie was "average", then it is likely that the active user might also think the movie was "average". Hence, for the training set, it is possible for the system to predict how the active user would rate each movie. For each set of weights (each individual), we can compute the predict rating of each movie i which active user A and user j had seen in training dataset as follow:

$$\text{PredictRate(A,i)} = \text{mean}RateA + \frac{1}{totalSimilar} * \Sigma(similar(A,j) * (Rate(j,i) - meanRatej))$$

where:

meanRateA: the average rating for all movies user A had seen

meanRate j: the average rating for all movies user j had seen

Rate(j,i) : actual rating of user j for movie item i

$$similar(A,j) = \frac{1}{Euclidean(A,j)}$$

totalSimilar : the sum of all Euclidean(A,j), where A and j (for all user j ≠ A in the neighborhood set) had seen common movie i

After computing each PredictRate(A,i) , we further define fitness value which is the differences between the actual and predicted ratings. Note that actual rating of user A for movie item i is known.

$$fitness = |\, predict\ vote(A,i) - \ actual\ rating(A,i)\,|$$

Then the fitness value of each set of weights (each individual) is the average of all fitness for all common movies which active user A and users in the neighborhood set had seen. So each individual has its fitness value and genetic algorithm will continue until termination condition.

2.3 Recommendation and Testing stage

In testing stage, we want to test whether the set of weights found in training stage has good performance for the movie active user A had not seen. The main concept is that: we compute the PredictRate of the active user A for the movie user A had not seen but user in the neighborhood set had seen. We can use the following equation which is mentioned before. After we get the predicted rating of active user A for the movie which A had not seen, we can compare it to the actual rating which we actually don't know but can get from testing dataset. Then we can get the error between predicted rating and actual rating. Note that this work doesn't use any actual rating of testing dataset to compute the predicted rating (see Fig. 5).

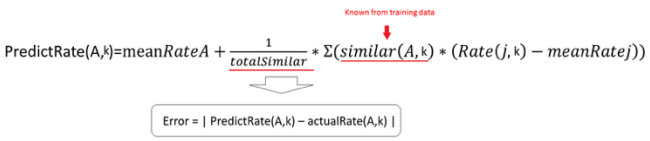


Figure 5. Testing stage flow chart.

2.4 Parallelize with OpenMP

We use Java language to implement whole serial version system. And we use omp4j (OpenMP for Java) to parallel the serial program.

3 EXPERIMENTAL METHODOLOGY

Fig. 6 shows our system environment. We run our program with 2 threads, 4 threads, 8 threads, 16 threads and compare their performance. We also use different generations and different population size to analyze their performance in different thread numbers. Our population size is 100 and generation number is 30 at the beginning. We test 100, 200, 400, 800 four population sizes with same generation (30 generations). And we test 30, 60, 90, 120 four different generations with same populations size (population size is 100).

```
kevin@kevin-BM1AF-BP1AF-BM6AF:~$ lscpu
Architecture:          x86_64
CPU 作業模式：    32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                4
On-line CPU(s) list:   0-3
每核心執行緒數：1
每通訊端核心數：4
Socket(s):             1
NUMA 節點：         1
供應商識別號：  GenuineIntel
CPU 家族：          6
型號：              60
Model name:            Intel(R) Core(TM) i5-4590 CPU @ 3.30GHz
製程：              3
CPU MHz：             1594.441
CPU max MHz:           3700.0000
CPU min MHz:           800.0000
BogoMIPS:              6584.85
虛擬：              VT-x
L1d 快取：          32K
L1i 快取：          32K
L2 快取：           256K
L3 快取：           6144K
NUMA node0 CPU(s):     0-3
```

Figure 6: system environment

4 EXPERIMENTAL RESULTS

4.1 Recommendation system output

We construct training model and use genetic algorithm to find good set of weights and similar Neighborhood of active user A. Now the recommendation system will recommend movie to active user and enter testing stage to test how good the set of weights in training stage found. Fig. 7 shows part of testing stage result. There are some movies which active user A had not seen but users in similar neighborhood had seen, those movies can compute Predicted rating (predict vote) and get actual rating (actual vote). And the error between those two value is shown at the end. Note that we show the movie ID instead of actual movie name.

```
Testing movie 26 ,predict vote: 3.9277777777777776 , actual vote : 2 | 1.9277777777777776
Testing movie 27 ,predict vote: 3.523931623931624 , actual vote : 4 | -0.476068376068376815
Testing movie 28 ,predict vote: 4.927777777777777 , actual vote : 5 | -0.07222222222222285
Testing movie 29 ,predict vote: 4.831623931623931 , actual vote : 4 | 0.8316239316239313
Testing movie 30 ,predict vote: 3.8067421126832977 , actual vote : 5 | -1.1932578873167023
Testing movie 31 ,predict vote: 2.5151598739893972 , actual vote : 2 | 0.5151598739893721
Testing movie 32 ,predict vote: 4.116689726315446 , actual vote : 5 | -0.883310273684554
Testing movie 33 ,predict vote: 3.976435647993886 , actual vote : 5 | -1.023564352006114
Testing movie 34 ,predict vote: 4.177777777777777 , actual vote : 4 | 0.17777777777777715
Testing movie 35 ,predict vote: 2.2277777777777774 , actual vote : 1 | 1.2277777777777774
Testing movie 36 ,predict vote: 4.127617596150074 , actual vote : 4 | 0.12761759615007406
Testing movie 37 ,predict vote: 4.511111111111111 , actual vote : 4 | 0.5111111111111111
Testing movie 38 ,predict vote: 3.8316239316239313 , actual vote : 3 | 0.8316239316239313
Testing movie 39 ,predict vote: 3.5531435107303566 , actual vote : 5 | -1.4468564892696434
Testing movie 40 ,predict vote: 1.036656427552325 , actual vote : 3 | -1.963343572447675
Testing movie 41 ,predict vote: 2.9515873015873013 , actual vote : 3 | -0.04841269841269685
Testing movie 42 ,predict vote: 4.511111111111111 , actual vote : 5 | -0.4888888888888893
Testing movie 43 ,predict vote: 3.165582655826558 , actual vote : 4 | -0.8344173441734419
Testing movie 44 ,predict vote: 2.5111111111111106 , actual vote : 5 | -2.4888888888888894
Testing movie 45 ,predict vote: 5.165582655826558 , actual vote : 5 | 0.16558265582655807
Testing movie 46 ,predict vote: 4.852962392677638 , actual vote : 5 | -0.14703760732236226
Testing movie 47 ,predict vote: 3.677088854588039 , actual vote : 5 | -1.3229114541196099
Testing movie 48 ,predict vote: 4.831623931623931 , actual vote : 4 | 0.8316239316239313
Testing movie 49 ,predict vote: 3.8082125603864734 , actual vote : 5 | -1.1917874396135266
Testing movie 50 ,predict vote: 4.677777777777777 , actual vote : 5 | -0.32222222222222285
```

Figure 7: testing stage result

Finally, the movies which the recommendation system should recommend to active user is shown in Fig. 8. Note

that our recommendation system only recommends 10 movies currently.

```
We recommend you following movie which you may like :
0. Career Girls (1997)
1. City of Lost Children, The (1995)
2. Evita (1996)
3. Return of the Jedi (1983)
4. Kiss the Girls (1997)
5. Kolya (1996)
6. Fargo (1996)
7. Seven Years in Tibet (1997)
8. Full Monty, The (1997)
9. Schindler's List (1993)
```

Figure 8: recommend movie

The current result shows that the average error value is about 0.8935, it is about 18% error rate in predicted rating and actual rating. We think that we can get more features from users and movies or modify genetic algorithm parameters to improve performance. Note that our goal doesn't focus on predicting high accurate rating. If we find those movies which have 5 rating value from similar user in neighborhood, even if the average error value is 0.89 the actual rating which we don't know will not be so poor. And we expect those movies which active user at least don't dislike or hate them.

4.2 Performance analysis

We divide our program into four parts : I/O, preprocessing, GA (run genetic algorithm), remaining(recommendation and testing stage). Fig. 9 shows serial time distribution of those parts. And we find that the program use almost all of time running GA and preprocessing, so we use OpenMP to parallel those two parts.

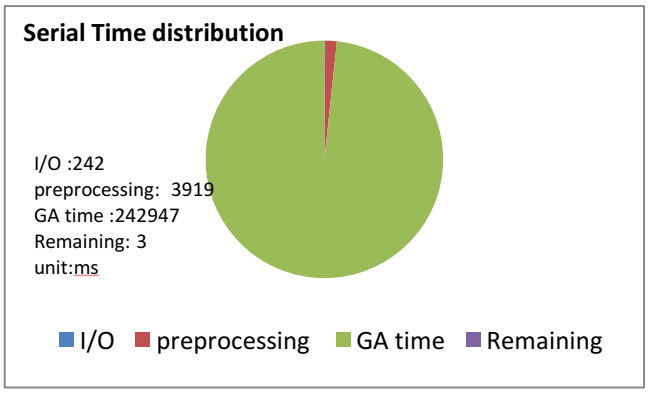
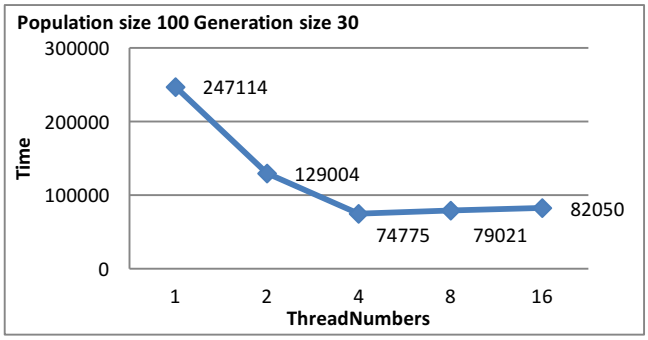


Figure 9: serial time distribution

At first, we run our program in population size 100 and 30 generations then compare the execution time and speed up with different thread numbers.(See Fig. 10) Because our environment only have four CPUs and each CPU only can run one thread, we have good speed up in thread number 1, 2, 4.



2 Threads	4 Threads	8 Thread
1.915	3.304	3.127

Speed up

Figure 10: execution time and speed up

Then we do two different experiments as follows:

1. Dynamic population size with fix generations
2. Dynamic generation with fix population size

Fig. 11 and Fig. 12 shows the result, 4 threads and 8 threads have similar execution time cause of the environment limit. And 1, 2, 4 threads performance is still good enough.

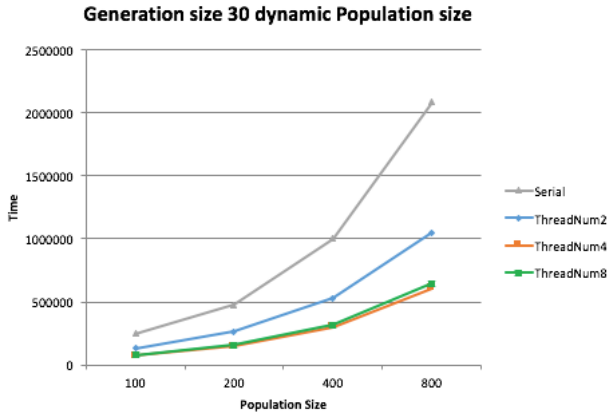


Figure 11: generation size 30 dynamic population size

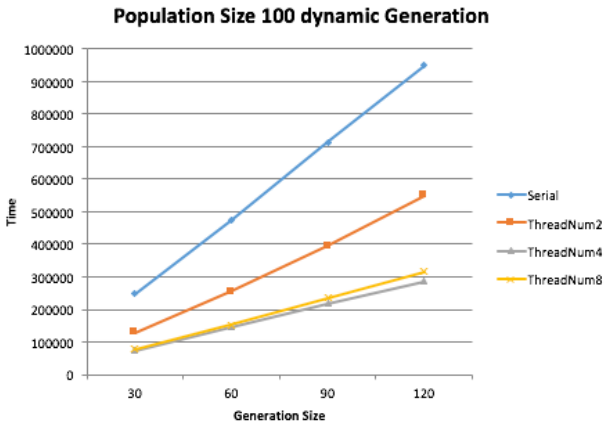


Figure 12: population size 100 dynamic generation

5 Conclusion

Our method has shown how evolutionary algorithm can help to implement a recommendation system. This was achieved by reformulating the problem of making recommendations into a supervised learning task. After all works our system find good set of weights and similar neighborhood then the system can recommend movies according to those information to active user. According to our experiment, we use omp4j to parallel the whole program and we find that scalability of our parallel work is well. It make our recommendation system more efficient. In future, we want to improve performance continuously. There are some targets as follows:

1. Get more features from users and movies
2. Modify genetic algorithm parameters
3. Change the diffi,f(A,j) definition
4. Change the computation of Euclidean and fitness value
5. Use CUDA to parallel
6. Use larger dataset to test performance

And we want to provide another way which the recommendation system recommends movies to active user. The system recommends movies which have high predicted rating (actual rating is not known) of active user instead of movies those users in neighborhood like. If the error rate is very low, we have high confidence in those movies active user may really like (the actual rating is very close to predicted rating).

REFERENCES

[1] Chein-Shung Hwang, Yi-Ching Su, Kuo-Cheng Tseng. 2010. Using Genetic Algorithms for Personalized Recommendation.

[2] Supiya Uijjin and Peter J. Bentley. 2002. Learning User Preferences Using Evolution.

[3] Schafer,J.B., Konstan, J.A. and Riedl, J. January2001.E-Commerce Recommendation Applications. Journal of Data Mining and Knowledge Discovery.

[4] Schafer,J.B., Konstan, J. and Riedl,J. 1999. Recommender Systems in E-Commerce. Proc. of the ACM 1999 Conf. on Electronic Commerce.

[5] Breese,J.S., Heckerman, D. and Kadie, C. 1998. Empirical analysis of predictive algorithms for collaborative filtering. In Proc. of the 14th Conf. on Uncertainty in AI, pp. 43-52.

[6] Herlocker, J.L., Konstan, J.A. & Riedl, J. 2000. Explaining Collaborative Filtering Recommendations. Proc. of ACM 2000 Conf. on Computer Supported Cooperative Work.

[7] omp4j tutorial : thttp://www.omp4j.org/