Particle Swarm Optimization Recommender System

Supiya Ujjin and Peter J. Bentley
University College London
Department of Computer Science
Gower Street, London WC1E 6BT
s.ujjin@cs.ucl.ac.uk

Abstract – Recommender systems are new types of internet-based software tools, designed to help users find their way through today's complex on-line shops and entertainment websites. This paper describes a new recommender system, which employs a particle swarm optimization (PSO) algorithm to learn personal preferences of users and provide tailored suggestions. Experiments are carried out to observe the performance of the system and results are compared to those obtained from the genetic algorithm (GA) recommender system [1] and a standard, non-adaptive system based on the Pearson algorithm [2].

I. INTRODUCTION

Decision-making is an integral part of everyday life. When faced with a dilemma, most of us are likely to gather some relevant information before making an informed decision. This information may come from various sources - known facts, predefined rules, opinions or even just gut feeling.

Many general lifestyle activities such as shopping for clothes, eating out and going to the cinema are highly dependent on personal tastes. It is this "personal" aspect that makes them much more difficult to recommend as they have a greater dependency on the person's personality. Therefore, people often seek advice from their friends, who can make suggestions based on their relationship and familiarity. However, there will come a time when friends cannot help due to the large number of choices available.

Recommender systems provide one way of circumventing this problem. As the name suggests, their task is to recommend or suggest items or products to the customer based on his/her preferences. These systems are often used by E-commerce websites as marketing tools to increase revenue by presenting products that the customer is likely to buy. An internet site using a recommender system can exploit knowledge of customers' likes and dislikes to build an understanding of their individual needs and thereby increase customer loyalty [3, 4].

This paper focuses on the use of particle swarm optimization (PSO) algorithm to fine-tune a profile-matching algorithm within a recommender system, tailoring it to the preferences of individual users. This enables the recommender system to make more accurate predictions of users' likes and dislikes, and hence better recommendations to users. The PSO technique was invented by Eberhart and Kennedy in 1995 and inspired by behaviours of social animals such as bird flocking or fish schooling. The algorithm itself is simple and involves adjusting a few parameters. With little modification, it can be applied to a wide range of applications. Because of this, PSO has received growing interest from researchers in various fields.

The paper is organised as follows: section II outlines related work, and section III describes the recommender system and PSO algorithm. Section IV provides experimental results and analysis. Finally section V concludes.

II. BACKGROUND

A. Recommender Systems

From the literature, it seems that the definition of the term "recommender system" varies depending on the author. Some researchers use the concepts: "recommender system", "collaborative filtering" and "social filtering" interchangeably [2]. Conversely, others regard "recommender system" as a generic descriptor that represents various recommendation/prediction techniques including collaborative, social, and content based filtering, Bayesian networks and association rules [5,6]. In this paper, we adopt the latter definition when referring to recommender systems.

MovieLens (http://www.movielens.umn.edu), a well-known research movie recommendation website, makes use of collaborative filtering technology to make its suggestions. This technology captures user preferences to build a profile by asking the user to rate movies. It searches for similar profiles (i.e., users that share the same or similar taste) and uses them to generate new suggestions (The dataset collected through the MovieLens website has been made available for research purposes and is used to test the PSO recommender system.)

By contrast, LIBRA (http://www.cs.utexas.edu/users/libra) combines a content-based approach with machine learning to make book recommendations. The content-based approach differs from collaborative filtering in that it analyses the contents of the items being recommended. Furthermore, each user is treated individually - there is no sense of "community" which forms the basis of collaborative filtering.

Dooyoo (http://www.dooyoo.co.uk) operates in a slightly different way. It too is a useful resource that provides recommendations to those seeking advice, but it focuses mainly on gathering qualitative opinions from its users, and then making them available to others. Visitors will often submit reviews on items or services ranging from health spas to mobile phones. These items are categorised in a similar fashion to the layout on a structured search engine, such as Yahoo!

124

Researchers at the University of the West of England have also been working on a movie Recommender System [8]. Their idea is to use the immune system to tackle the problem of preference matching and recommendation. User preferences are treated as a pool of antibodies and the active user is the antigen. The difference in their approach and the other existing methods is that they are not interested in finding the one best match but a diverse set of antibodies that are a close match.

The Genetic Algorithm (GA) Recommender System used in the experiments is part of the previous work. More detailed descriptions can be found in [1]. The system follows the same structure as the PSO recommender system, discussed later in section III. The only difference is that the GA system used a genetic algorithm to evolve feature weights for the active user (in this paper a PSO algorithm is used for the same purpose). An individual in the population represented a possible solution which in this case was a set of feature weights that defines the active user's preference. A fitness function using a modified Euclidean distance function was employed. Experimental results showed that the GA system performed very well compared to a non-adaptive approach based on the Pearson algorithm.

B. Particle Swarm Optimization

Although particle swarm optimization is a population-based evolutionary technique like genetic algorithms, it differs in that each particle or solution contains a position, velocity and acceleration. The velocity and acceleration are responsible for changing the position of the particle to explore the space of all possible solutions, instead of using existing solutions to reproduce [9]. As particles move around the space, they sample different locations. Each location has a fitness value according to how good it is at satisfying the objective, in this case, defining the user's preferences. Because of the rules governing the swarming process, particles will eventually swarm around the area in the space where fittest solutions are. PSO is becoming popular for many diverse applications, for example:

Van den Bergh et. al applied the original Particle Swarm Optimisation as well as the Cooperative Particle Swarm Optimiser (a variation of it) to the task of training neural networks [12]. Particle swarms were used to find the optimal weights of a Product Unit Neural Network. The Cooperative Optimiser employs multiple swarms where each swarm handles a part of the vector being optimised. The *split factor*, a parameter added to the original PSO algorithm, determines the number of constituent parts for each particle that is also the number of swarms used.

Swarm Music is an interactive music improviser by Blackwell and Bentley that employs a swarm of particles which are interpreted as musical events. These particles interact with each other according to rules that are based on flocking and swarming models. The music space has dimensions that represent musical parameters such as pulse, pitch and loudness. The swarm is attracted to the targets which are external musical events that are captured and placed in the space. As the particles move around the music

space of adjustable parameters, improvisations are produced interactively with external musicians [9].

III. SYSTEM OVERVIEW

The system described in this paper is based around a collaborative filtering approach, building up profiles of users and then using an algorithm to find profiles similar to the current user. (In this paper, we refer to the current user as the *active user, A*). Selected data from those profiles are then used to build recommendations. Because profiles contain many attributes, many of which have sparse or incomplete data [7], the task of finding appropriate similarities is often difficult. To overcome these problems, current systems (such as MovieLens) use stochastic and heuristic-based models to speed up and improve the quality of profile matching. This work takes such ideas one step further, by applying a particle swarm optimization algorithm to the problem of profile matching.

In this research, the MovieLens dataset was used for initial experiments. It contains details of 943 users, each with many parameters or *features*: demographic information such as age, gender and occupation is collected when a new user registers on the system. Every time a vote is submitted by a user, it is recorded in the database with a timestamp. The movie information in the dataset includes genres, and theatre and video release dates. Both the PSO and GA recommender systems use 22 features from this data set: 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.

A. Profile Generator

Before recommendations can be made, the movie data must first be processed into separate profiles, one for each person, defining that person's movie preferences.

A profile for user j, denoted $profile(j)$, is represented as an array of 22 values for the 22 features considered. The profile has two parts: a variable part (the rating value, which changes according to the movie item being considered at the time), and a fixed part (the other 21 values, which are only retrieved once at the beginning of the program). Because user j may have rated many different movies, we define $profile(j,i)$ to mean the profile for user j on movie item i, see fig. 1.

1	2	3	4	..22
Rating	Age	Gender	Occupation	18 Genre frequencies
5	23	0	45	000000100010000000

Figure 1: *profile(j,i)* - profile for user *j* with rating on movie item *i*, if *i* has a rating of 5.

Once profiles are built, the process of recommendation can begin. Given an active user A, a set of profiles similar to $profile(A)$ must be found.

B. Neighbourhood Selection

The success of a collaborative filtering system is highly dependent upon the effectiveness of the algorithm in finding

125

the set or neighbourhood of profiles that are most similar to that of the active user. It is vital that, for a particular neighbourhood method, only the best or closest profiles are chosen and used to generate new recommendations for the user. There is little tolerance for inaccurate or irrelevant predictions.

The neighbourhood selection algorithm consists of three main tasks: *Profile Selection*, *Profile Matching* and *Best Profile Collection*.

1) Profile Selection

In an ideal world, the entire database of profiles would be used to select the best possible profiles. However this is not always a feasible option, especially when the dataset is very large or if resources are not available. As a result, most systems opt for random sampling and this process is the responsibility of the profile selection part of the algorithm.

This work investigates two methods of profile selection:

1. Fixed: the first *n* users from the database are always used in every experiment

2. Random: *n* users are picked randomly from the database,

where $n = 10$ or 50 in our experiments.

2) Profile Matching

After profile selection, the profile matching process then computes the distance or similarity between the selected profiles and the active user's profile using a distance function. This research focuses on this profile matching task, i.e., the PSO algorithm is used to fine-tune profile matching for each active user.

From the analysis of Breese et. al [2], it seems that most current recommender systems use standard algorithms that consider only "voting information" as the feature on which the comparison between two profiles is made. However in real life, the way in which two people are said to be similar is not based solely on whether they have complimentary opinions on a specific subject, e.g., movie ratings, but also on other factors, such as their background and personal details. If we apply this to the profile matcher, issues such as demographic and lifestyle information which include user's age, gender and preferences of movie genres must also be taken into account. Every user places a different importance or priority on each feature. These priorities can be quantified or enumerated. Here we refer to these as *feature weights*. For example, if a male user prefers to be given recommendations based on the opinions of other men, then his feature weight for gender would be higher than other features. In order to implement a truly personalised recommender system, these weights need to be captured and fine-tuned to reflect each user's preference. Our approach shows how such weights can be found using a particle optimization technique.

A potential solution to the problem of fine-tuning feature weights, $w(A)$, for the active user, A is represented as a set of weights as shown below in Figure 2.

w_1	w_2	w_3	$\ldots$	w_{22}

Figure 2: A set of weights representing a user's preference.

where w_f is the weight associated with feature f. Each set contains 22 weighting values, representing a position of a particle in 22-dimensional search space. As particles move around the space, these values are continuously adjusted (described in section D) in order to find the best particle with a set of weights which accurately describes the active user's preference.

The comparison between two profiles can now be conducted using a modified Euclidean distance function, which takes into account multiple features. *Euclidean(A,j)* is the similarity 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 is the active user

j is a user provided by the profile selection process, where $j \neq A$

z is the number of common movies that users A and j have rated.

w_f, is the active user's weight for feature f

i is a common movie item, where *profile(A,i)* and *profile(j,i)* exists.

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

Note that before this calculation is made, the profile values are normalised to ensure they lie between 0 and 1. When the weight for any feature is zero, that feature is ignored. This way we enable feature selection to be adaptive to each user's preferences. The difference in profile values for occupation is either 0, if the two users have the same occupation or 1 otherwise.
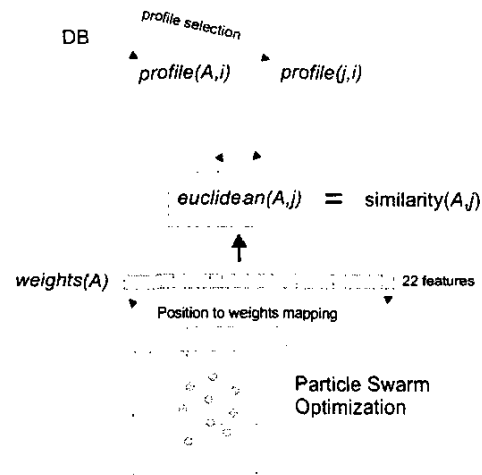

Figure 3: Calculating the similarity between A and j.

3) Best Profile Collection

Once the Euclidean distances, *euclidean(A,j)*, have been found between *profile(A)* and *profile(j)* for all values of *j*

126

picked by the profile selection process, the "best profile collection" algorithm is called. This ranks every *profile(j)* according to its similarity to *profile(A)*. The system then simply selects the users whose Euclidean distance is above a certain threshold value (considered most similar to the active user) as the neighbourhood of *A*. This value is a system constant that can be changed.

C. Making Recommendation

To make a recommendation, given an active user *A* and a neighbourhood set of similar profiles to *A*, it is necessary to find movie items seen (and liked) by the users in the neighbourhood set that the active user has not seen. These are then presented to the active user through a user interface. Because the neighbourhood set contains those users who are most similar to *A* (using in our case the specific preferences of *A* through attained weighting values), movies that these users like have a reasonable probability of being liked by *A*.

D. PSO Algorithm

The PSO algorithm has been used to attain feature weights for the active user, and hence help tailor the matching function to the user's specific personality and tastes.

1) Particle Dynamics

A conventional PSO algorithm [10] with the velocity clamping rule introduced in [9] was chosen. The PSO has 22-dimensional space, each axis has values ranging from 0 to 255 (corresponding to the simple unsigned binary genetic encoding with 8 bits for each of the 22 genes, used in the implementation of the genetic algorithm (GA) recommender system in the Background section). As mentioned above, each particle has a position and a velocity (their values are randomised initially). The position with the highest fitness score (the fitness function is described below) in each iteration is set to be the entire swarm's global best (gbest) position, towards which other particles move. In addition, each particle keeps its best position that it has visited, known as the particle's personal best (pbest). The particle dynamics are governed by the following rules which update particle positions and velocities:

$$v_i = wv_i + c_1r_1(x_{pbest,i} - x_i) + c_2r_2(x_{gbest} - x_i)$$

$$if(|v_i| > v_{max}) \quad v_i = (v_{max}/|v_i|)v_i$$

$$x_i = x_i + v_i$$

where

x_i is the current position of particle i

x_{pbest} is the best position attained by particle i

x_{gbest} is the swarm's global best position

v_i is the velocity of particle i

w is a random inertia weight between 0.5 and 1

c_1 and c_2 are spring constants whose values are set to 1.494 [10]

r_1 and r_2 are random numbers between 0 and 1

2) Fitness Function

Calculating the fitness for this application is not trivial. A set of feature weights can be calculated from the position

values. First, the importance of the 18 genre frequencies are reduced by a given factor, the *weight reduction size*. This is done because the 18 genres can be considered different categories of a single larger feature, Genre. Reducing the effect of these weights is therefore intended to give the other unrelated features (movie rating, age, gender, occupation) a more equal chance of being used. Second, the total value of the position is then calculated by summing the position values on all 22 axes. Finally, the weighting value for each feature can be found by dividing the real value by the total value. The sum of all the weights will then add up to unity.

Every particle's current position (set of weights) in the swarm must be employed by the profile matching processes within the recommender system. So the recommender system must be re-run on the MovieLens dataset for each new position, in order to calculate its fitness.

It was decided to reformulate the problem as a supervised learning task. As described previously, given the active user *A* and a set of neighbouring profiles, recommendations for *A* can be made. In addition to these recommendations, it is possible to predict what *A* might think of them. For example, if 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 MovieLens dataset, it was possible for the system to both recommend new movies and to predict how the active user would rate each movie, should he go and see it.
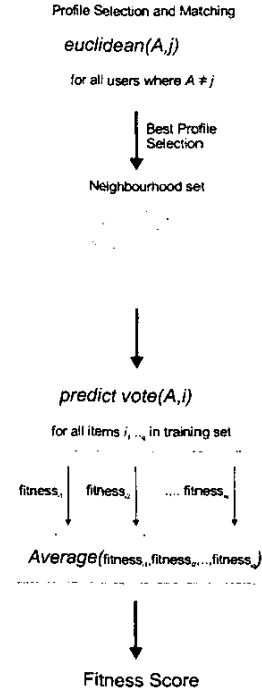

Figure 4: finding the fitness score of a particle's current position (the active user's feature weights).

The predicted vote computation used in this paper has been taken from [2] and modified such that the Euclidean distance function (*Profile Matching* section) now replaces the weight

127

in the original equation. The predicted vote, *predict_vote(A,i)*, for *A* on item *i*, can be defined as:

$$predict\_vote(A,i) = mean_A + k \sum_{j=1}^{n} euclidean(A,j)(vote(j,i) - mean_j)$$

where: mean*j* is the mean vote for user *j*

k is a normalising factor such that the sum of the euclidean distances is equal to 1.

vote(j,i) is the actual vote that user *j* has given on item *i*

n is the size of the neighbourhood.

All the movie items that the active user has seen are randomly partitioned into two datasets: a training set (1/3) and a test set (2/3). To calculate a fitness measure for a particle's current position, the recommender system finds a set of neighbourhood profiles for the active user, as described in *Neighbourhood Selection* section. The ratings of the users in the neighbourhood set are then employed to compute the predicted rating for the active user on each movie item in the training set. Because the active user has already rated the movie items, it is possible to compare the actual rating with the predicted rating. So, the average of the differences between the actual and predicted votes of all items in the training set are used as fitness score to guide future swarming process, see figure 4.

IV. EXPERIMENTS

Four sets of experiments were designed to observe the difference in performance between the PSO, GA [1] and a standard, non-adaptive recommender systems based on the Pearson algorithm [2]. In each set of experiments, the predicted votes of all the movie items in the test set (the items that the active user has rated but were not used in fitness evaluation) were computed using the final feature weights for that run. These votes were then compared against those produced from the simple Pearson algorithm and the GA system.

The four experiments evaluated two system variables to assess their effect on system performance: the *profile selection* task (the way in which profiles were selected from the database), and the size of the neighbourhood.

The four sets of experiments were as follows:

Experiment 1: Each of the first 10 users was picked as the active user in turn, and the first 10 users (fixed) were used to provide recommendations.

Experiment 2: Each of the first 50 users was picked as the active user in turn, and the first 50 users (fixed) were used to provide recommendations.

Experiment 3: Each of the first 10 users was picked as the active user in turn, and 10 out of 944 users were picked randomly and used to provide recommendations (the same 10 used per run for all three systems).

Experiment 4: Each of the first 50 users was picked as the active user in turn, and 50 out of 944 users were picked randomly and used to provide recommendations (the same 50 used per run for all three systems).

These were performed on two versions of the system:

Zero tolerance – the accuracy of the system is found by calculating the percentage of the number of ratings that the system predicted correctly out of the total number of available ratings by the current active user. The results for experiments 1 to 4 with zero tolerance are shown in figures 5 to 8, respectively.

At-Most-One tolerance – same as zero tolerance but if the difference between the predicted and actual ratings is less than or equal to 1 then this predicted rating is considered to be correct. This takes into account the vulnerability of the rating system – ratings on an item can vary depending on many factors such as the time of the day and the user's mood at this time. For example, if the system predicts 4 out of 5 for an item and the actual rating from the user is 5, the prediction still shows that the user demonstrates a strong preference towards the item. Figures 9 to 12 show the results for experiments 1 to 4 with at-most-one tolerance.

The following parameter values were kept the same in all four experiments:

---

1. *swarm/population size* = 75. The number of particles/individuals in the swarm/population at each generation.

2. *maximum number of iterations for each run* = 300. If the number of iterations reaches this value and the solution has not been found, the best solution for that iteration is used as the final result.

3. *weight reduction size* = 4. The scaling factor for the 18 genre frequencies.

4. *number of runs* = 30. The number of times the system was run for each active user.

The Pearson algorithm (PA) used in the experiments is based on the *k* Nearest Neighbour algorithm. A correlation coefficient, shown below, is used as the matching function for selecting the *k* users that are most similar to the active user to give predictions. This replaces the *Euclidean* function described earlier; all other details remain the same.

$$correlation(A,j) = \frac{\sum_{i=1}^{z}(vote(A,i) - mean_A)(vote(j,i) - mean_j)}{\sqrt{\sum_{i=1}^{z}(vote(A,i) - mean_A)^2(vote(j,i) - mean_j)^2}}$$

The Genetic Algorithm Recommender System used an elitist genetic algorithm, where a quarter of the best individuals in the population were kept for the next generation. When creating a new generation, individuals were selected randomly out of the top 40% of the whole population to be parents. Two offspring were produced from every pair of parents, using single-point crossover with probability 1.0. Mutation was applied to each locus in genotype with probability 0.01. A simple unsigned binary genetic encoding was used in the implementation, using 8 bits for each of the 22 genes which represented the 22 features considered.
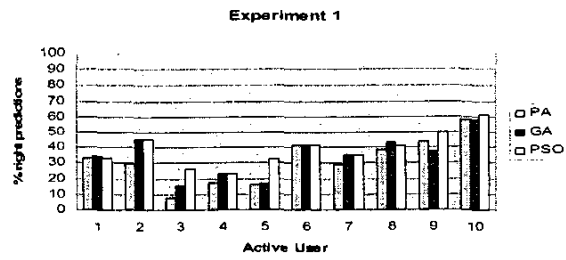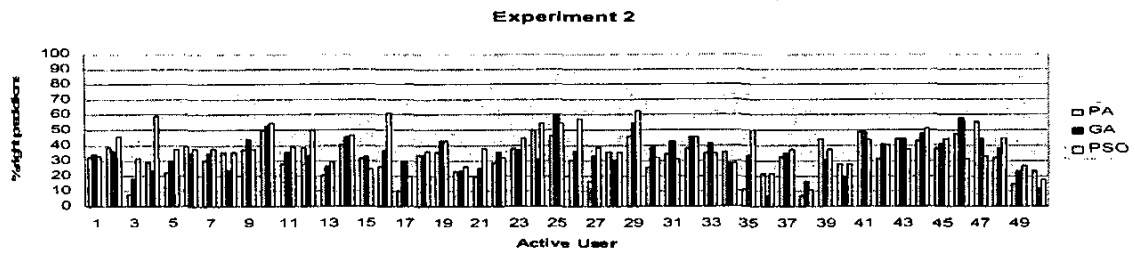
Experiment 1

Figure 5: Results for experiment 1 – Zero tolerance

Experiment 2

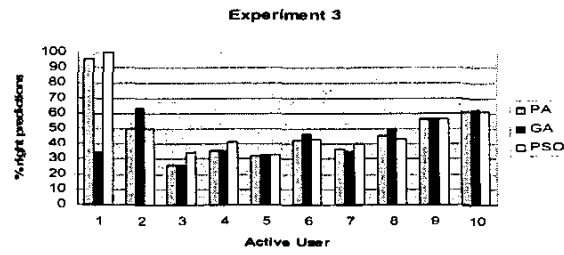Figure 6: Results for experiment 2 – Zero tolerance

Experiment 3

Figure 7: Results for experiment 3 – Zero tolerance
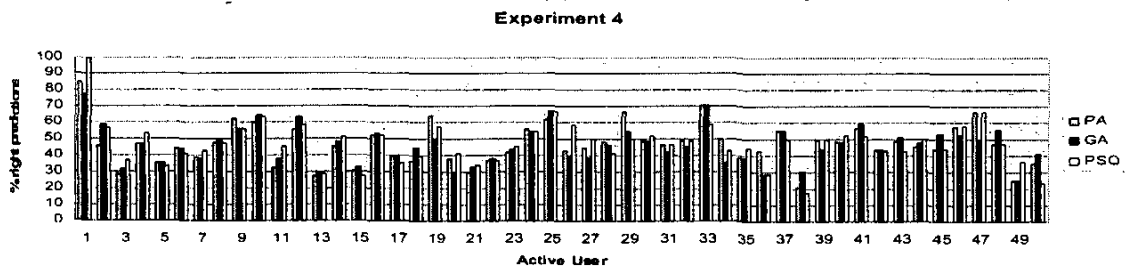
Experiment 4

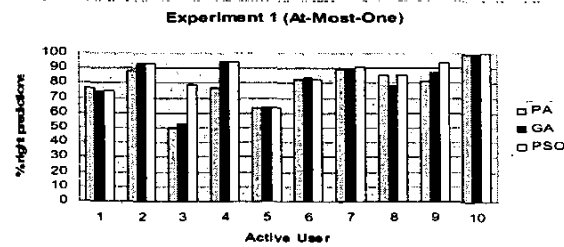Figure 8: Results for experiment 4 – Zero tolerance

Experiment 1 (At-Most-One)

Figure 9: Results for experiment 1 – At-Most-One tolerance

129

Experiment 2 (At-Most-One)
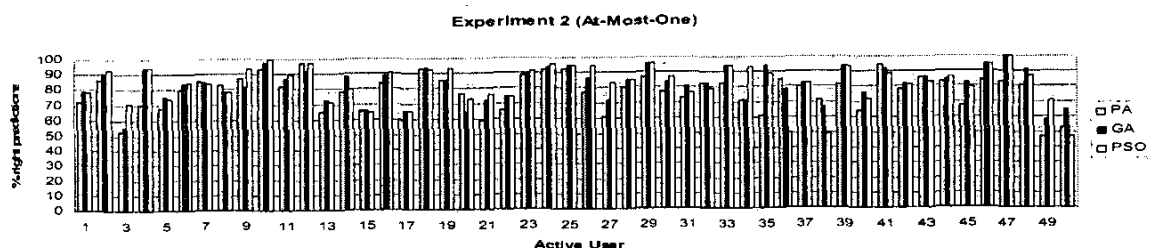

Figure 10: Results for experiment 2 – At-Most-One tolerance

Experiment 3 (At-Most-One)
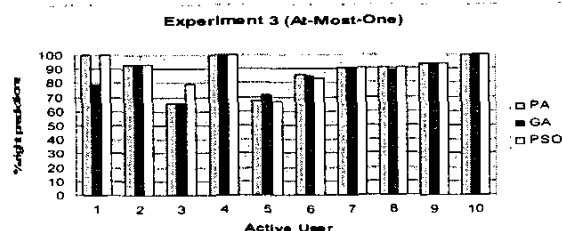
Figure 11: Results for experiment 3 – At-Most-One tolerance

Experiment 4 (At-Most-One)
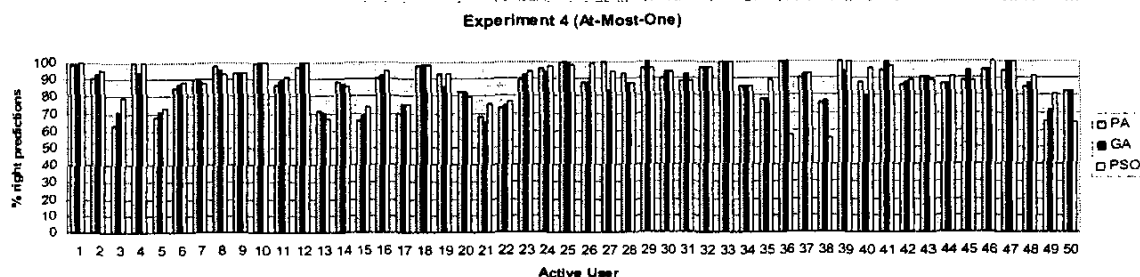
Figure 12: Results for experiment 4 – At-Most-One tolerance
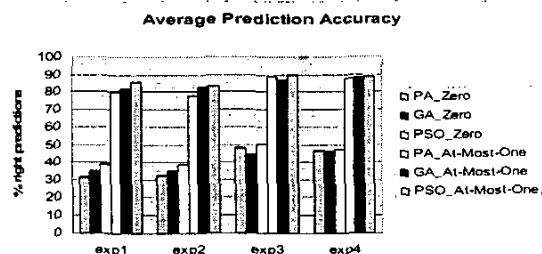
Average Prediction Accuracy

Figure 13: Average Prediction Accuracy

A. Results

Whilst the predictions computed with the Pearson algorithm always remain the same given the same parameter values, those obtained from the PSO and GA vary according to the feature weights of that run. Out of the 30 runs for each active user in each experiment, the run with the best feature weights (that gave the highest percentage of right predictions) was chosen and plotted against the result from the Pearson algorithm.[1]

By looking at the results obtained from the experiments, the following observations have been found:

• On the whole the PSO performed significantly well compared to the other two systems, see figures 5 to 8 for results obtained from experiments 1 to 4.

• When random profile selection was used, in most cases the prediction accuracy for PSO rose considerably and outperformed the other two systems. Figures 7 and 8 show the results for experiments 3 and 4.

• The performance of the PSO system improved greatly from being 40-50% accurate with zero tolerance level to 60-100% when at-most-one tolerance level was employed. Figure 13 shows average prediction accuracy for all four experiments with both zero and at-most-one tolerance levels.

---

[1] The best rather than average was plotted since this is closest to the real world scenario where this system could be run off-line and the current best set of feature weights would be set as the initial preference of the active user.

• The speed of execution of the PSO system was over 10% faster than that of the GA.

B. Analysis of Results

It was found that the PSO performed very well compared to the other two systems for all four experiments with both Zero and At-Most-One tolerance levels. However, the observation found in [1] that as the number of users goes up, the probability of finding a better matched profile should be higher and hence the accuracy of the predictions should increase, still applies to the GA system here, but is not the case for the PSO recommender. One explanation to this fall in the average accuracy level as the number of users increases is that originally the users that were selected to be part of the neighbourhood to give recommendations are highly similar to the active user. As the number of users increases, more users are being considered and this could sometimes result in many less similar users being added to the neighbourhood and hence, lower overall prediction accuracy. This problem will be examined in more detail and tackled in future work where we treat the users themselves as particles in the swarm.

Results for the PSO from experiments 3 and 4 confirm the observation found in the last published work on the GA recommender [1] that random sampling is better than fixing which users to select. This is because it allows the search to consider a greater variety of profiles.

In addition, average and worst results were examined and compared among the three methods. It was found that PSO still achieved the best performance in most experiments compared to the GA and PA systems. The only case when PSO did not achieve the highest performance was when worst results were considered in experiment 2, emphasising the problem of increased number of users mentioned earlier. However, this problem does not have any effect on the performance of PSO in experiment 4 when random sampling was used.

As mentioned earlier, only the run(s) with the best feature weights for each active user were considered for this analysis. We now look into these runs in more detail to see how the feature weights obtained and users selected for the neighbourhood in these runs played a part in determining user preference. In the GA recommender, when more than 1 run for an active user achieved the same best performance (highest number of votes being predicted correctly) results indicate that the same set of users had been selected for the neighbourhood to give recommendations [1]. However, this is not the case for the PSO system: many runs that attained the same best performance did not select the same set of users. By looking at the history of particle paths, when more than 1 particle attained the same best performance, the global best position was picked randomly from one of these particles and it is this position that other particles move towards. It is possible that the other "best" particles that were neglected could have contained better solution had they been given the chance for the swarm to explore their surrounding space. Future work will explore this in more detail by adding another rule to the swarming process making the

particles move towards the central location of all "best" particles.

A duplicate user to active user 1 was inserted into a run for active user 1 in experiment 3 and two runs in experiment 4 in order to observe how good the system was at using the information from this duplicate user to give recommendations. The results demonstrate the best run was obtained from the runs which contain the duplicate user and that the PSO picked this to be the only user in the neighbourhood resulting in 100% prediction accuracy for the active user.

V. CONCLUSIONS

This work has shown how particle swarm optimization can be employed to fine-tune a profile-matching algorithm within a recommender system, tailoring it to the preferences of individual users. Experiments demonstrated that the PSO system outperformed a non-adaptive approach and obtained higher prediction accuracy than the Genetic Algorithm system and Pearson algorithm in most cases. In addition, compared to the GA approach, the PSO algorithm achieved the final solution significantly faster, making it a more efficient way of improving performance where computation speed plays an important part in recommender systems.

References

[1] Ujjin, S. and Bentley, P. J. 2002. Learning User Preferences Using Evolution. In Proceedings of the 4th Asia-Pacific Conference on Simulated Evolution and Learning. Singapore.

[2] Breese, J.S., Heckerman, D. and Kadie, C. 1998. Empirical analysis of predictive algorithms for collaborative filtering. In Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence, pp. 43-52.

[3] Schafer, J. B., Konstan, J. A. and Riedl, J. January 2001. 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. Proceedings of the ACM 1999 Conference on Electronic Commerce.

[5] Terveen, L. and Hill, W. 2001. Beyond Recommender Systems: Helping People Help Each Other. In HCI In The New Millenium, Carroll, J. ed. Addison-Wesley.

[6] Delgado, J.A. February 2000. Agent-Based Information Filtering and Recommender Systems on the Internet. PhD thesis, Nagoya Institute of Technology.

[7] Herlocker, J.L., Konstan, J. A. and Riedl, J. 2000. Explaining Collaborative Filtering Recommendations. Proceedings of the ACM 2000 Conference on Computer Supported Cooperative Work.

[8] Cayzer, S. and Aickelin U. 2001. A Recommender System based on the Immune Network. Submitted and under review.

[9] Blackwell, T.M. and Bentley, P.J. 2002. Dynamic Search with Charged Swarms. In Proceedings of the Genetic and Evolutionary Computation Conference 2002. New York.

[10] Eberhart, R.C. and Shi, Y. 2001. Particle Swarm Optimization: Developments, Applications and Resources. In Proceedings of the 2001 Congress on Evolutionary Computation. vol.1, pp.81-86.

[11] Bentley, P. J. and Corne, D. W. 2001. Creative Evolutionary Systems. Morgan Kaufman Pub.

[12] van den Bergh, F. and Engelbrecht, A.P. 2001. Training Product Unit Networks using Cooperative Particle Swarm Optimisers. In Proceedings of IJCNN 2001, Washington DC.