Improving Capabilities of Recommender Systems Using Swarm Intelligence

Supiya Ujjin

A dissertation submitted for the degree of

Doctor of Philosophy

of the

University of London

Department of Computer Science
University College London
Abstract

This thesis investigates the use of novel genetic and swarm intelligence algorithms to increase performance and capabilities of recommender systems. Through the use of these algorithms, the work introduces the concept of recommender systems adapting to the user's recommendation preferences to provide more accurate and useful suggestions. An extensive literature review evaluates all related areas of research and reveals that such systems do not exist. The development of two generic recommender systems, one using a genetic algorithm (GA) and the other using particle swarm optimisation (PSO), are then presented.

The thesis describes a number of significant advances. Firstly, a new data representation of a user profile is introduced, which takes into account multiple features of the data. Secondly, similarity measures are investigated in order for user profiles to be compared accurately. It is vital that an appropriate measure is used as the success of any recommender system based on a collaborative filtering approach depends significantly on this. Thirdly, a new fitness function is devised, which uses the quality of recommendations to guide evolution, by reformulating the problem of making recommendations into a supervised learning task.

Additionally, the thesis describes significant advances in the field of swarm intelligence during the development of the second system. Two novel swarming algorithms were created. The ClusterPSO algorithm is the first application of swarm intelligence to the problem of adaptive recommender systems. The ClusterWeight then builds on the ClusterPSO algorithm and is the first to simultaneously cluster and search for solutions. The idea of dynamic swarming is introduced, which allows users themselves to form or join a group of similar users, enabling real-time open-ended adaptation to continuously changing data.

The prediction accuracy, speed and usability (in terms of relevance of recommendations and user scalability) of the systems are assessed by performing comparisons with existing algorithms used within recommender systems and through a pilot study. The results of these experiments show that capabilities of recommender systems can be improved by the use of evolutionary algorithms and swarm intelligence.
Acknowledgements

There are many people that I would like to thank for a huge variety of reasons.

Firstly, I would like to express my deep and sincere gratitude to my supervisor and friend, Peter Bentley, for his continual support and guidance throughout the course of my Ph.D. His constant encouragement and enthusiasm have helped me get through difficult periods and keep me on course. Without his input and tireless efforts, this journey would have been far less tolerable.

I would also like to thank my other supervisor, Philip Treleaven, for providing me with the opportunity to work on this research and for his constructive criticism and advice throughout.

The financial support of the Thai government is gratefully acknowledged.

I am very grateful to my special friends at UCL: Jungwon Kim, Sanjeev Kumar, Bill Langdon and Anthony Ruto who stood by me in my time of need, for making my Ph.D. experience an enjoyable and memorable one, for their much-valued feedback and for proof-reading chapters of this thesis. I am especially indebted to Sanjeev and Anthony for being at the other end of the telephone regardless of the time of day. A special thanks to Akira Sato for his work on the fashion avatar. I also extend my thanks to others at UCL who have assisted me in some way or another.

I am particularly thankful to my girlie friends: Duanpen Lertpibulpanya, Selina Man, Jaratsri Rungrattanaubol, Chanchana Thanachayanont, Poovadee Tuchinda and Yaowarez Yongsunthon for the day-long chats, their words of encouragement and advice, their understanding and, most of all, for their priceless friendship.

Many thanks to the IC lot: Richard Apoolingum, Shishani Batal, Simon Bone, Bintal Patel, Michael Dean, Ben Jefferys and Peter Steele for dragging me away from my work when I needed a break, for the Monday night cinema outings and for being great friends. I would also like to thank Ben for introducing me to the field of collaborative filtering. A
big thank you to Tony Wong for all his help and support and for proof-reading chapters in this thesis.

A special thanks to other wonderful friends: Vanee Caseechetty, Samuel Crawley, Max Christian, Andreas Giannopoulos, Joanna Joseph, Henrik Karlsson, Nima Kaveh, Ling Li, Peter Montienvichienchai, Yumiko Nakayama, Swany Ratanadirek, Sophie Sabatie, Carrie Scrivener, Joe Simpson, Avy Tahan, Siu Wong and those already mentioned for participating in the pilot study and for letting me make use of your personal data.

A special big thank you to David Yoo-Foo for putting a smile on my face when I was feeling down, for his words of encouragement when I wanted to give up, for his understanding when I had to work at weekends, for keeping me sane, for believing in me, for creating custom graphing tools, for setting up the website for the pilot study, for proof-reading every chapter of this thesis and, most of all, for always being there for me.

My heartfelt thanks and appreciation go to my family, without whom it would have been impossible to finish this thesis. A big thank you to my sisters, Kunteera and Jayanud Ujjin, for their endless encouragement and for putting up with and taking care of me. A special thanks to my aunts, Piyada, Pinida and Piyamarn Lewchalamwong, for their love and emotional support and for not letting me quit. Finally, a huge thank you to my parents, Prasit and Piyawan Ujjin, for their unconditional love, support and encouragement - this work is dedicated to them.
Table of Contents

CHAPTER 1: Introduction ................................................................. 22
  1.1. Motivation ............................................................................ 22
  1.2. Thesis Hypothesis ............................................................... 25
  1.3. Thesis Objectives ................................................................. 25
  1.4. Contributions ...................................................................... 26
  1.5. Publications .......................................................................... 28
  1.6. Thesis Structure ................................................................. 28

CHAPTER 2: Literature Review ....................................................... 31
  2.1. Introduction ........................................................................... 31
  2.2. Recommender Systems ......................................................... 31
      2.2.1. Collaborative Filtering .................................................. 32
            Active Collaborative Filtering ........................................... 35
            Passive Collaborative Filtering ......................................... 36
            Automated Collaborative Filtering .................................... 37
            Related Work on Movie Recommender Systems ............... 38
            Evaluation Metrics ......................................................... 40
            Shortcomings of Collaborative Filtering Systems .............. 42
      2.2.2. Alternative Techniques ................................................ 44
            Content-based Filtering .................................................. 45
            Classification ................................................................. 46
            Clustering ...................................................................... 47
            Bayesian Networks ........................................................ 51
            Association Rules .......................................................... 52
            Immune Systems .......................................................... 52
      2.2.3. Summary of Recommender Systems .............................. 54
  2.3. Adaptive Systems ............................................................. 55
      2.3.1. Evolutionary Algorithms .............................................. 55
            Genetic Algorithms (GAs) ............................................... 56
      2.3.2. Swarm Intelligence ..................................................... 59
            Flocks ......................................................................... 61
            Particle Swarms ........................................................... 62
            Ant Colony ..................................................................... 71
2.3.3. Summary of Adaptive Systems ................................................................. 73
2.4. Summary ............................................................................................................ 73

CHAPTER 3: Evolving Good Recommendations with a Genetic Algorithm .......... 75
3.1. MovieLens Dataset .......................................................................................... 75
3.1.1. Composition of Data .................................................................................. 77
3.1.2. Preparation of Training and Test Data ....................................................... 79
3.2. GA Recommender System ............................................................................. 80
3.2.1. Profile Generator ....................................................................................... 82
3.2.2. Neighbourhood Selection ......................................................................... 83
  Profile Selection .................................................................................................. 83
  Profile Matching ................................................................................................. 83
  Best Profile Collection ..................................................................................... 85
3.2.3. Making a Recommendation ...................................................................... 86
3.2.4. Genetic Algorithm .................................................................................... 86
  Fitness function ................................................................................................. 87
3.3. Experiments ..................................................................................................... 89
3.3.1. Experimental Setup ................................................................................... 91
3.3.2. Results ........................................................................................................ 92
3.3.3. Analysis of Results .................................................................................... 94
3.4. Summary ......................................................................................................... 103

CHAPTER 4: A Recommender System using Particle Swarm Optimisation .......... 104
4.1. Comparison between PSO, GA and PA Recommender Systems ................. 104
4.1.1. Flying Geese PSO (FG_PSO) Recommender ........................................... 104
  Particles ........................................................................................................... 105
  Particle Dynamics ............................................................................................. 106
  Experiments for Flying Geese PSO ............................................................... 108
  Results for Flying Geese PSO ........................................................................ 110
  Analysis of Results for Flying Geese PSO ................................................... 115
4.2. Variations of PSO Recommender .................................................................. 116
4.2.1. Variation 1: Quad Feature PSO (QF_PSO) ............................................. 116
  Experiments 1 for Variation 1 ........................................................................ 117
  Results 1 for Variation 1 .................................................................................. 117
  Experiments 2 for Variation 1 ........................................................................ 119
  Results 2 for Variation 1 .................................................................................. 120
4.2.2. Variation 2: Migrating PSO (M_PSO) ...................................................... 120
4.2.3. Variation 3: Flocking PSO (F_PSO) .......................................................... 124
Experiments for Variation 3 ................................................................................. 125
Results for Variation 3 .......................................................................................... 125
4.3. Analysis of Flocking PSO.............................................................. 129
4.3.1. Software Analysis Tools ............................................................................. 129
Swarm’s Prediction Accuracy Tool ..................................................................... 129
Graphical Data Display Tool ............................................................................... 130
3D Visualisation Tool ........................................................................................... 130
4.3.2. Swarm Behaviours ....................................................................................... 133
Behaviour 1 – No Convergence with Multiple Solutions ................................134
Behaviour 2 – No Solution ................................................................................. 135
Behaviour 3 – Good Convergence with Best Solution ..................................... 137
Behaviour 4 – Premature Convergence with Undesirable Solution ................. 138
Behaviour 5 – Slow Convergence with Good Solution .................................... 140
Behaviour 6 – Convergence to Good Area .......................................................... 142
4.3.3. A Variation of Flocking PSO ...................................................................... 146
Experiment for a variation of Flocking PSO ...................................................... 146
Results for a variation of Flocking PSO ............................................................. 146
4.4. Summary .............................................................................................................. 151

CHAPTER 5: ClusterPSO: A Novel Swarming Algorithm ............................................. 152
5.1. ClusterPSO Recommender................................................................. 153
5.1.1. Swarming Space ........................................................................................... 153
Bounded Space ..................................................................................................... 153
Unbounded Space ................................................................................................. 154
5.1.2. Users as Particles ......................................................................................... 155
5.1.3. Neighbourhood of Friends .......................................................................... 156
Best Friends ........................................................................................................... 156
Outsiders ................................................................................................................ 157
5.1.4. Neighbourhood Selection .......................................................................... 157
(1) Check for Neighbourhood ............................................................................ 159
(2) Compute Similarities ...................................................................................... 160
(3) Update Neighbourhood Best \( (nbest) \) ......................................................... 161
(4) Update Best Friend \( (pbest) \) ......................................................................... 161
7.3. Future Work ........................................................................................................228

7.3.1. Improvements to existing ClusterWeight system .....................................228

Suggestion 1: Feedback Integration ....................................................................228

Suggestion 2: Recommendations from Celebrities ............................................229

Suggestion 3: Film Clubs .....................................................................................230

Suggestion 4: Interactions with Avatar ...............................................................230

Suggestion 5: Secondary Friends .........................................................................231

Suggestion 6: Confidence Display .......................................................................231

7.3.2. Alternative machine learning task and intelligent cluster .......................232

7.3.3. Incorporating lifestyle data .........................................................................233

7.3.4. Applied Psychology .....................................................................................233

Brand awareness .................................................................................................234

Multi-cultural consumer behaviour ..................................................................235

Bibliography ...........................................................................................................236

Appendix A: Standard Deviation Values for the Pearson Algorithm, Genetic
Algorithm and Particle Swarm Optimisation Recommender Systems ..........254

Appendix B: Experiments with 4 system variables for ClusterPSO.....................257

B.1. Swarming Space Size ..................................................................................257

B.2. Maximum Range of Neighbourhood ...........................................................262

B.3. Maximum Velocity .......................................................................................269

B.4. Repulsive Factor ..........................................................................................274

Appendix C: Experiments with system variables for ClusterWeight ....................281

C.1. Maximum Range ..........................................................................................281

C.2. Maximum Velocity .......................................................................................287

Appendix D: Online Implementation ....................................................................292

D.1. Pilot Study ....................................................................................................292

D.1.1. Users ........................................................................................................292

D.1.2. Experiments .............................................................................................297

D.1.3. Results .....................................................................................................298

D.1.4. User Feedback ..........................................................................................300

D.1.5. Analysis of Results ..................................................................................303

D.1.6. Experiments with Explicit Feedback ......................................................307

D.1.7. Online Recommender System Architecture .........................................309

Appendix E: Preliminary Work on Fashion and Sizing Recommendations ..........312

E.1. Fashion Advisor ..........................................................................................312
E.2. Sizing Advisor.................................................................313
E.3. Real vs. Virtual Advisors..................................................315
E.4. User Interface (Avatar).....................................................317
List of Figures

Figure 2.1: A decision tree which predicts whether an individual watched movie C. The bar charts show the probabilities of 'watched' and 'not watched' for C, depending on parent movies, A and B. ................................................................. 52
Figure 2.2: A network produced by AIS, taken from work by Timmis (2001b) ............ 53
Figure 2.3: The simple genetic algorithm ........................................................................ 57
Figure 2.4: 1-point crossover at position 5 ................................................................. 59
Figure 2.5: A boid's neighbourhood ............................................................................. 62
Figure 2.6: The lbest topology where k = 2 ................................................................ 64
Figure 2.7: The gbest topology .................................................................................. 65
Figure 2.8: PSO algorithm ......................................................................................... 66
Figure 2.9: The top diagram shows two ants leaving the nest at the same time. The bottom diagram shows that each ant has taken a different route (the pheromone trails are represented by the dotted lines). The ant that took the shorter path returns first and the amount of pheromones left on this path is now twice as much. Thus, other ants are more attracted to this shorter path than the longer one. ........................................................................ 71
Figure 3.1: Relationships and dependencies between the various elements in the database ........................................................................................................................................... 76
Figure 3.2: Gender split .............................................................................................. 77
Figure 3.3: Age groups ............................................................................................... 78
Figure 3.4: Genres ....................................................................................................... 78
Figure 3.5: Occupations ............................................................................................... 79
Figure 3.6: The number of training and test items for each active user .................... 80
Figure 3.7: The algorithm describing the GA Recommender System ....................... 81
Figure 3.8: profile(j,i) - profile for user j with rating on movie item i, if i has a rating of 5. ........................................................................................................................................... 82
Figure 3.9: Phenotype of an individual in the population ............................................. 84
Figure 3.10: Calculating the similarity between A and j ........................................... 85
Figure 3.11: Finding the fitness score of an individual (the active user's feature weights). ........................................................................................................................................... 89
Figure 3.12: Comparison between PA and GA - best results for experiment 1 ....... 92
Figure 3.13: Comparison between PA and GA – best results for experiment 2 ......... 92
Figure 3.14: Comparison between PA and GA – best results for experiment 3 ......... 93
Figure 3.15: Comparison between PA and GA - best results for experiment 4 ..........93
Figure 3.16: Comparison between PA and GA - average results for experiment 1 ........96
Figure 3.17: Comparison between PA and GA - average results for experiment 2 .........96
Figure 3.18: Comparison between PA and GA - average results for experiment 3 ........97
Figure 3.19: Comparison between PA and GA - average results for experiment 4 ........97
Figure 3.20: Comparison between PA and GA - worst results for experiment 1 ..........98
Figure 3.21: Comparison between PA and GA - worst results for experiment 2 ...........98
Figure 3.22: Comparison between PA and GA - worst results for experiment 3 ..........99
Figure 3.23: Comparison between PA and GA - worst results for experiment 4 ..........99
Figure 3.24: Feature weights from best runs for active user 2 (weights 5 to 22 are lower because of the scaling factor) ..........................................................100
Figure 3.25: Feature weights from the best run for active user 8 .................................100
Figure 3.26: Feature weights for active user 7 ............................................................101
Figure 3.27: Feature weights for active user 1 in 4-feature experiment .........................102
Figure 4.1: Calculating the similarity between the active user A and a user j ...............105
Figure 4.2: The algorithm describing the Flying Geese PSO Recommender System ....107
Figure 4.3: Comparison between PA, GA and PSO - best results for experiment 1 with zero tolerance ............................................................................................................110
Figure 4.4: Comparison between PA, GA and PSO - best results for experiment 2 with zero tolerance ............................................................................................................110
Figure 4.5: Comparison between PA, GA and PSO - best results for experiment 3 with zero tolerance ............................................................................................................111
Figure 4.6: Comparison between PA, GA and PSO - best results for experiment 4 with zero tolerance ............................................................................................................111
Figure 4.7: Comparison between PA, GA and PSO - best results for experiment 1 with at-most-one tolerance .................................................................111
Figure 4.8: Comparison between PA, GA and PSO - best results for experiment 2 with at-most-one tolerance .................................................................111
Figure 4.9: Comparison between PA, GA and PSO - best results for experiment 3 with at-most-one tolerance .................................................................112
Figure 4.10: Comparison between PA, GA and PSO - best results for experiment 4 with at-most-one tolerance .................................................................112
Figure 4.11: Average accuracy of PA, GA and PSO with zero and at-most-one tolerance levels .................................................................................................113
Figure 4.12: Comparison between QF_PSO and FG_PSO - best results for experiment 1 with zero tolerance. ..........................................................................................................................117
Figure 4.13: Comparison between QF_PSO and FG_PSO - best results for experiment 2 with zero tolerance. ..........................................................................................................................118
Figure 4.14: Comparison between QF_PSO and FG_PSO - best results for experiment 3 with zero tolerance. ..........................................................................................................................118
Figure 4.15: Comparison between QF_PSO and FG_PSO - best results for experiment 4 with zero tolerance. ..........................................................................................................................118
Figure 4.16: Average accuracy of QF_PSO and FG_PSO with zero and at-most-one tolerance levels ..................................................................................................................................................119
Figure 4.17: Comparison between 3 different weight reduction sizes in FG_PSO for experiment 1 ..................................................................................................................................................120
Figure 4.18: Modified pbest and gbest update method for M_PSO (replace loop 7 in figure 4.2) ..................................................................................................................................................121
Figure 4.19: Comparison between FG_PSO and M_PSO - best results for experiment 1 with zero tolerance. ..................................................................................................................................................122
Figure 4.20: Comparison between FG_PSO and M_PSO - best results for experiment 2 with zero tolerance. ..................................................................................................................................................122
Figure 4.21: Comparison between FG_PSO and M_PSO - best results for experiment 3 with zero tolerance. ..................................................................................................................................................122
Figure 4.22: Comparison between FG_PSO and M_PSO - best results for experiment 4 with zero tolerance. ..................................................................................................................................................123
Figure 4.23: Average accuracy of FG_PSO and M_PSO with zero and at-most-one tolerance levels ..................................................................................................................................................123
Figure 4.24: pbest positions at final iteration ..................................................................................................................................................................................................................................................123
Figure 4.25: Modified gbest update method for Flocking PSO (replace loop 7 in figure 4.2) ..................................................................................................................................................125
Figure 4.26: Comparison between FG_PSO and F_PSO - best results for experiment 1 with zero tolerance. ..................................................................................................................................................125
Figure 4.27: Comparison between FG_PSO and F_PSO - best results for experiment 2 with zero tolerance. ..................................................................................................................................................126
Figure 4.28: Comparison between FG_PSO and F_PSO - best results for experiment 3 with zero tolerance. ..................................................................................................................................................126
Figure 4.29: Comparison between FG_PSO and F_PSO - best results for experiment 4 with zero tolerance. ..................................................................................................................................................126
Figure 4.30: Average accuracy of FG_PSO and F_PSO with zero and at-most-one tolerance levels ................................................................. 127
Figure 4.31: Global best movement for user 2 (run 5) in experiment 4 ..................................... 127
Figure 4.32: The gbest selection process for Flocking and Flying Geese PSO .................... 128
Figure 4.33: Swarm’s Prediction Accuracy Tool ................................................................. 130
Figure 4.34: Example outputs from the analysis tools ...................................................... 131
Figure 4.35: Behaviour 1 example ..................................................................................... 135
Figure 4.36: Behaviour 2 example ..................................................................................... 136
Figure 4.37: Behaviour 3 example ..................................................................................... 138
Figure 4.38: Behaviour 4 example ..................................................................................... 139
Figure 4.39: Behaviour 5 example ..................................................................................... 141
Figure 4.40: Behaviour 6 example ..................................................................................... 143
Figure 4.41: Example movement of Migrating PSO ......................................................... 144
Figure 4.42: Distribution of behaviours for experiment 1 .................................................. 144
Figure 4.43: Distribution of behaviours for experiment 2 .................................................. 145
Figure 4.44: Distribution of behaviours for experiment 3 .................................................. 145
Figure 4.45: Distribution of behaviours for experiment 4 .................................................. 145
Figure 4.46: Comparison between 2 different maximum velocities in F_PSO - best results for experiment 1 with zero tolerance ................................................................. 147
Figure 4.47: Comparison between 2 different maximum velocities in F_PSO - best results for experiment 2 with zero tolerance ................................................................. 147
Figure 4.48: Comparison between 2 different maximum velocities in F_PSO - best results for experiment 3 with zero tolerance ................................................................. 147
Figure 4.49: Comparison between 2 different maximum velocities in F_PSO - best results for experiment 4 with zero tolerance ................................................................. 148
Figure 4.50: Average accuracy of F_PSO and F_PSO_255 with zero and at-most-one tolerance levels .............................................................................................................. 148
Figure 4.51: Distribution of behaviours for experiment 1 .................................................. 149
Figure 4.52: Distribution of behaviours for experiment 2 .................................................. 149
Figure 4.53: Distribution of behaviours for experiment 3 .................................................. 149
Figure 4.54: Distribution of behaviours for experiment 4 .................................................. 150
Figure 5.1: Bounded swarming space ................................................................................. 154
Figure 5.2: Unbounded space in a shape of a donut ............................................................ 154
Figure 5.3: Unbounded space displayed in 2 dimensions .................................................... 155
Figure 5.4: The process of neighbourhood selection for the active user, A. ....................... 158
Figure 5.5: The neighbourhood of the active user, $A$ ......................................................... 159
Figure 5.6: Check for Neighbourhood – special case ......................................................... 160
Figure 5.7: Attraction to one best friend ............................................................................. 162
Figure 5.8: Attraction to multiple best friends ............................................................... 162
Figure 5.9: Repulsive Force with one outsider ............................................................... 163
Figure 5.10: Repulsive Force with multiple users ............................................................... 163
Figure 5.11: The algorithm describing the ClusterPSO recommender system ............ 164
Figure 5.12: User preference graph with 10 users ............................................................... 168
Figure 5.13: An example of possible neighbourhoods ......................................................... 168
Figure 5.14: User preference graph with 50 users ............................................................... 170
Figure 5.15: COP-KMEANS algorithm .............................................................................. 171
Figure 5.16: Comparison between COP_KMEANS and Flocking PSO - best results for experiment 1 with zero tolerance. ......................................................... 172
Figure 5.17: Comparison between COP_KMEANS and Flocking PSO - best results for experiment 1 with at-most-one tolerance. ......................................................... 172
Figure 5.18: Comparison between COP_KMEANS and Flocking PSO - best results for experiment 2 with zero tolerance. ......................................................... 172
Figure 5.19: Comparison between COP_KMEANS and Flocking PSO - best results for experiment 2 with at-most-one tolerance. ......................................................... 173
Figure 5.20: Comparison between PA and ClusterPSO - best results for both algorithms for experiment 1 with zero tolerance ......................................................... 178
Figure 5.21: Comparison between PA and ClusterPSO - best results for both algorithms for experiment 2 with zero tolerance ......................................................... 179
Figure 5.22: Comparison between GA and ClusterPSO - best results for both algorithms for experiment 1 with zero tolerance ......................................................... 179
Figure 5.23: Comparison between GA and ClusterPSO - best results for both algorithms for experiment 2 with zero tolerance ......................................................... 179
Figure 5.24: Comparison between Flocking PSO and ClusterPSO - best results for both algorithms for experiment 1 with zero tolerance ......................................................... 180
Figure 5.25: Comparison between Flocking PSO and ClusterPSO - best results for both algorithms for experiment 2 with zero tolerance ......................................................... 180
Figure 5.26: Comparison between COP_KMEANS and ClusterPSO - best results for both algorithms for experiment 1 with zero tolerance ......................................................... 181
Figure 5.27: Comparison between COP_KMEANS and ClusterPSO - best results for both algorithms for experiment 2 with zero tolerance ......................................................... 181
Figure 6.1: Modified ClusterPSO algorithm ................................................................. 185
Figure 6.2: The active user $A$ is attracted to $B$ who is both $A$'s best friend and
neighbourhood best ........................................................................................................... 188
Figure 6.3: $A$ is attracted to both $B$ and his best location ........................................... 188
Figure 6.4: Comparison between ClusterPSO and Modified ClusterPSO - best results for
experiment 1 with zero tolerance ..................................................................................... 190
Figure 6.5: Visualisation of modified ClusterPSO (swarming space = 1500) .................. 190
Figure 6.6: Ripple effect of users chasing after one another ......................................... 191
Figure 6.7: Comparison between 2 swarming space sizes - best results for experiment 1
........................................................................................................................................... 192
Figure 6.8: Visualisation of modified ClusterPSO with swarming space = 7500 ............ 192
Figure 6.9: 10 vs. 50 users ............................................................................................... 193
Figure 6.10: Visualisation of 50 users ............................................................................... 194
Figure 6.11: ClusterWeight algorithm ............................................................................ 195
Figure 6.12: Best results for experiment 1 with at-most-1 tolerance ............................... 198
Figure 6.13: Best results for experiment 2 with at-most-1 tolerance ............................... 198
Figure 6.14: Comparison between Pearson Algorithm and ClusterWeight - best results for
both algorithms for experiment 1 with zero tolerance ................................................... 199
Figure 6.15: Comparison between Pearson Algorithm and ClusterWeight - best results for
both algorithms for experiment 2 with zero tolerance ................................................... 200
Figure 6.16: Comparison between Pearson Algorithm and ClusterWeight - worst results
for both algorithms for experiment 1 with zero tolerance ............................................. 200
Figure 6.17: Comparison between Pearson Algorithm and ClusterWeight - worst results
for both algorithms for experiment 2 with zero tolerance ............................................. 200
Figure 6.18: Comparison between Pearson Algorithm and ClusterWeight - average results
for both algorithms for experiment 1 with zero tolerance ............................................. 201
Figure 6.19: Comparison between Pearson Algorithm and ClusterWeight - average results
for both algorithms for experiment 2 with zero tolerance ............................................. 201
Figure 6.20: Comparison between GA and ClusterWeight - best results for both
algorithms for experiment 1 with zero tolerance .......................................................... 204
Figure 6.21: Comparison between GA and ClusterWeight - best results for both
algorithms for experiment 2 with zero tolerance .......................................................... 204
Figure 6.22: Comparison between GA and ClusterWeight - worst results for both
algorithms for experiment 1 with zero tolerance .......................................................... 205
Figure 6.23: Comparison between GA and ClusterWeight - worst results for both algorithms for experiment 2 with zero tolerance ........................................................205
Figure 6.24: Comparison between GA and ClusterWeight - average results for both algorithms for experiment 1 with zero tolerance ........................................................205
Figure 6.25: Comparison between GA and ClusterWeight - average results for both algorithms for experiment 2 with zero tolerance ........................................................206
Figure 6.26: Comparison between Flocking PSO and ClusterWeight - best results for both algorithms for experiment 1 with zero tolerance ........................................................209
Figure 6.27: Comparison between Flocking PSO and ClusterWeight - best results for both algorithms for experiment 2 with zero tolerance ........................................................209
Figure 6.28: Comparison between Flocking PSO and ClusterWeight - worst results for both algorithms for experiment 1 with zero tolerance ........................................................209
Figure 6.29: Comparison between Flocking PSO and ClusterWeight - worst results for both algorithms for experiment 2 with zero tolerance ........................................................210
Figure 6.30: Comparison between Flocking PSO and ClusterWeight - average results for both algorithms for experiment 1 with zero tolerance ........................................................210
Figure 6.31: Comparison between Flocking PSO and ClusterWeight - average results for both algorithms for experiment 2 with zero tolerance ........................................................210
Figure 6.32: Comparison between COP_KMEANS and ClusterWeight - best results for both algorithms for experiment 1 with zero tolerance ........................................................213
Figure 6.33: Comparison between COP_KMEANS and ClusterWeight - best results for both algorithms for experiment 2 with zero tolerance ........................................................213
Figure 6.34: Comparison between COP_KMEANS and ClusterWeight - worst results for both algorithms for experiment 1 with zero tolerance ........................................................213
Figure 6.35: Comparison between COP_KMEANS and ClusterWeight - worst results for both algorithms for experiment 2 with zero tolerance ........................................................214
Figure 6.36: Comparison between COP_KMEANS and ClusterWeight - average results for both algorithms for experiment 1 with zero tolerance ........................................................214
Figure 6.37: Comparison between COP_KMEANS and ClusterWeight - average results for both algorithms for experiment 2 with zero tolerance ........................................................214
Figure 7.1: Adjustable feature weights ................................................................................229
Figure 7.2: An approach for obtaining feedback on neighbourhood of friends ................229
Figure 7.3: Recommender System Avatar interface ................................................................231
Figure 7.4: Feature weights representing e-mail preferences .............................................232
Figure B.1: Comparison between experiments 1 to 4 - best results with swarming space size of 1500. ................................................................. 257
Figure B.2: Experiment 1 (10 users) ................................................................. 258
Figure B.3: Experiment 2 (50 users) ................................................................. 259
Figure B.4: Comparison for best results obtained for experiment 2 with 4 different swarming space sizes .......................................................... 260
Figure B.5: Experiment 2 - size 3 (7500 units) ................................................. 261
Figure B.6: Comparison of best results obtained from experiments 1 (space size of 1500) and 2 (space size of 7500). .................................................... 261
Figure B.7: Comparison of best results obtained from experiment 1 with 2 swarming space sizes of 1500 and 7500. .............................................. 262
Figure B.8: Comparison of best results obtained from experiment 1 with 4 different ranges ................................................................. 263
Figure B.9: Comparison of best results obtained from experiment 2 with 4 different ranges ................................................................. 263
Figure B.10: Comparison of best results obtained from experiment 3 with 4 different ranges ................................................................. 263
Figure B.11: Comparison of best results obtained from experiment 4 with 4 different ranges ................................................................. 264
Figure B.12: The visualisation of experiment 1 with different range values for run 12. 265
Figure B.13: The visualisation of experiment 2 with different range values for run 12. 265
Figure B.14: Experiment 1 - Average movement for run 12 ......................... 268
Figure B.15: Experiment 2 - Average movement for run 12 ......................... 268
Figure B.16: Comparison of best results obtained from experiment 1 with 3 different velocities ................................................................. 269
Figure B.17: Comparison of best results obtained from experiment 2 with 3 different velocities ................................................................. 269
Figure B.18: Comparison of best results obtained from experiment 3 with 3 different velocities ................................................................. 270
Figure B.19: Comparison of best results obtained from experiment 4 with 3 different velocities ................................................................. 270
Figure B.20: Experiment 1 - Average movement for run 2 ......................... 271
Figure B.21: Experiment 2 - Average movement for run 2 ......................... 272
Figure B.22: Experiment 1 – visualisation of run 1 ......................................... 273
Figure B.23: Experiment 2 – visualisation of run 1 ......................................... 273
Figure B.24: Comparison of best results obtained from experiment 1 with 3 different repulsive factors. .................................................................275
Figure B.25: Comparison of best results obtained from experiment 2 with 3 different repulsive factors. .................................................................275
Figure B.26: Comparison of best results obtained from experiment 3 with 3 different repulsive factors. .................................................................276
Figure B.27: Comparison of best results obtained from experiment 4 with 3 different repulsive factors. .................................................................276
Figure B.28: Experiment 1 - Average movement for run 2 ..............................................277
Figure B.29: Experiment 2 - Average movement for run 2 ..............................................278
Figure B.30: Experiment 1 – visualisation of run 1 ...........................................................278
Figure B.31: Experiment 2 – visualisation of run 1 ...........................................................279
Figure B.32: Comparison of best results obtained from experiment 1 for 3 different repulsive factors with maximum velocity of 200 .......................................................279
Figure B.33: Comparison of best results obtained from experiment 2 for 3 different repulsive factors with maximum velocity of 200 .......................................................280
Figure C.1: Comparison between different range values and the mean ratings - best results for experiment 1 with zero tolerance. ..............................................282
Figure C.2: Comparison between different range values and the mean ratings - best results for experiment 2 with zero tolerance. ..............................................282
Figure C.3: Visualisation of runs 9 (left) and 3 (right) for the maximum range of 1000 with 10 users ......................................................................................283
Figure C.4: Visualisation of run 3 for maximum range of 2000 displayed in 2 different angles – 1st and 2nd features (left) and 3rd and 4th features (right) ......................284
Figure C.5: Comparison between experiments 1 and 2 - best results with maximum range of 1000 .................................................................................285
Figure C.6: Comparison between experiments 1 and 2 - best results with maximum range of 2000 .................................................................................285
Figure C.7: Final feature weights for run 3 – range 2000 ..................................................286
Figure C.8: Final feature weights for users 10 and 4 .......................................................287
Figure C.9: Comparison between 2 velocity values - best results for experiment 1 ......287
Figure C.10: Comparison between 2 velocity values - best results for experiment 2 ....288
Figure C.11: Comparison between 2 velocity values - worst results for experiment 1 ....288
Figure C.12: Comparison between 2 velocity values - worst results for experiment 2 ...289
Figure C.13: Comparison between 2 velocity values - average results for experiment 1289
Figure C.14: Comparison between 2 velocity values - average results for experiment 2289
Figure C.15: Visualisation of a good run from velocity 400 (run 1), showing both final
neighbourhoods and best locations (left) and best locations alone (right) ..........290
Figure C.16: Visualisation of a converged run from velocity 400 (run 9), showing both
final neighbourhoods and best locations (left) and best locations alone (right).....291
Figure C.17: Visualisation of final best locations for maximum velocity of 200 (left) and
maximum velocity of 400 (right) ..............................................................................291
Figure D.1: The login page...................................................................................................293
Figure D.2: The first page after the user has successfully logged on ..............................293
Figure D.3: A list of movies beginning with ‘A’. The user can rate a movie by selecting a
drop-down list to the right of the movie and choose a rating between 1 (worst) and 5
(best). Clicking on movie title will open a new window with the movie information
from the Internet Movie Database .................................................................................293
Figure D.4: By clicking on ‘Your rated movies’, this page is displayed which lists the
currently rated movies for the user where ratings can be updated .............................294
Figure D.5: Main associations of users .............................................................................295
Figure D.6: User partitions based on age group ..............................................................295
Figure D.7: User partitions based on gender ..................................................................296
Figure D.8: User partitions based on occupation ............................................................296
Figure D.9: Best, average and worst results obtained from the pilot study with zero
tolerance ......................................................................................................................299
Figure D.10: Best, average and worst results obtained from the pilot study with at-most-
one tolerance ..............................................................................................................299
Figure D.11: Feature weights from the best run for Patrick .............................................304
Figure D.12: Feature weights from the best run for Pia ....................................................304
Figure D.13: Final neighbourhoods of Sabrina’s best run ..............................................305
Figure D.14: Feature weights from the best run for Scott ..............................................307
Figure D.15: Comparison between the original and after feedback – best results with zero
tolerance ......................................................................................................................308
Figure D.16: Comparison between the original and after feedback – best results with at-
most-one tolerance .................................................................................................309
Figure D.17: System Architecture ..................................................................................310
Figure D.18: All current positions on the server with possible Profile set for user A
encircled .......................................................................................................................311
List of Tables

Table 3.1: Table Descriptions .................................................................................................................77
Table 3.2: Experimental Parameters ..........................................................................................................90
Table 3.3: recommended movies for active user 1 .......................................................................................102
Table 4.1: Parameters values used in the experiments .............................................................................110
Table 5.1: Similarity table ..........................................................................................................................167
Table 5.2: User preference table for the first 10 users .............................................................................167
Table 5.3: User preference table for the first 50 users .............................................................................169
Table 5.4: Experimental Parameters ........................................................................................................177
Table 6.1: Experimental Parameters ........................................................................................................189
Table 7.1: Time taken by the GA, PSO, ClusterWeight, PA and COP_KMEANS systems to perform a run ......................................................................................................................227
Table A.1: Standard deviations for GA and PSO in experiment 1 ..............................................................254
Table A.2: Standard deviations for GA and PSO in experiment 2 ..............................................................255
Table A.3: Standard deviations for PA, GA and PSO in experiment 3 ........................................................255
Table A.4: Standard deviations for PA, GA and PSO in experiment 4 ........................................................256
Table B.1: Various swarming space sizes for experiment 2 .........................................................................259
Table B.2: Various range values ..................................................................................................................263
Table B.3: Various velocity values ...............................................................................................................269
Table B.4: Various repulsive factors ..........................................................................................................275
Table C.1: Experimental Parameters ........................................................................................................281
Table C.2: Best friend list for run 3 ..............................................................................................................286
Table D.1: Parameter values .........................................................................................................................297
Table D.2: A breakdown of the total number of movie items rated by each user ........................................298
Table D.3: recommendation list sent to Sabrina ..........................................................................................301
Table D.4: User feedback .............................................................................................................................301
Table D.5: Additional comments given by the users ..................................................................................303
Table E.1: Comparison between real and virtual advisors .........................................................................315
Table E.2: Virtual Fashion advisor survey .................................................................................................319
CHAPTER 1

Introduction

1.1. Motivation

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 the task of providing suitable suggestions (what clothes one should wear, food to eat and movies to watch) a difficult one as they have a great dependency on the person's personality. Therefore, people often seek advice from their friends or family members, who can make suggestions based on their relationship and familiarity. However, this is becoming more and more impractical due to the sheer number of choices available. A good example is one brought about by the rapid expansion of electronic commerce. There is a need for someone or something that can provide opinions like a friend or family member who knows you well, a sales assistant who is familiar with the range of products available, and/or perhaps an advisor who can cast an expert opinion.

A computer system that can take into account personal recommendation preferences and continuously fine-tune itself according to them is therefore needed in order to give more accurate recommendations that are considered useful to the user of the system. "Useful" may have different meanings in this context - for example, a person may choose to be adventurous and try new ideas. A person who normally likes action movies may want to watch a comedy and may therefore want to consult people who like comedy movies for suggestions.

As markets change and expand, the use of recommender systems looks set to increase dramatically. It is becoming general practice for large companies to gather information on
their customers by providing a form of reward card scheme which gives customers an incentive to participate. Detailed analysis can be performed on this data to give companies an insight into their customers' behaviour. Recently, several companies from different industries, such as supermarkets, mobile phone service providers and credit cards, have merged their reward schemes (for example, Nectar\(^1\)) which potentially may allow them to gain a broader understanding of their customers. Recommender systems technology is a perfect candidate to fully exploit this data.

Collaborative filtering has been one of the most widely used approaches in commercial recommender systems. The idea behind it is that people with similar tastes should be able to recommend items to each other. The work presented in this thesis is based on this technique as it is domain independent and can therefore have many different applications. However, the task of finding similar people is not trivial. Because the data that represent people may be sparse or incomplete, the task of finding appropriate similarities is often difficult. In addition to this, there is no concrete rule or solution to define similarity between two human beings. A typical example is movie recommendations where users are asked to rate movies that they have seen. It seems that most existing recommender systems use standard statistical-based algorithms that consider only "rating information" as the feature on which the comparison between two users is made (Breese et al. 1998). In other words, if person \(A\) rates a movie the same as person \(B\), then they are considered similar. However in real life, the way in which two people are said to be similar is not based solely on whether they have similar opinions on a specific subject, e.g., movie ratings, but also on other factors, such as their background and personal details. Thus, issues such as demographic and lifestyle information which include user's age, gender and movie genres should also be taken into account when similarities are being considered. Every user places a different importance or priority on each feature. These priorities can be quantified or enumerated. In this thesis, 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 preferences. Indeed, much of the work in this thesis investigates algorithms which can find suitable values for such feature weights.

\(^1\) http://www.nectar.com/
This method of capturing the personalities and priorities of users has several desirable characteristics

• By having a set of feature weights to represent personal recommendation preferences for each user, explanations behind each recommendation can be provided. This makes the traditional “black box” computation transparent to the users and thus, should increase the level of confidence that the users have in the system.
• As users change their preferences over time, the feature weights can dynamically adapt to reflect these changes.

One approach to capturing a user’s preferences is to evolve feature weights by use of a genetic algorithm (GA). GAs are traditionally known to be robust and produce good results for a wide range of applications. More importantly, they exhibit adaptivity. Details of a GA-based system and experiments carried out are presented in chapter 3.

Another approach is to use swarm intelligence (SI) – a new research field which has become increasingly popular. A swarming system comprises non-intelligent agents which work collectively to achieve a significant task. Using this idea, by making the task of finding similar users distributed, more potential solutions can be considered and hence, it is proposed that the accuracy of recommendations can be improved. Because agents run in parallel, the speed at which the recommendations are generated can also be increased. This is crucial to an online system as users expect the system to respond quickly to their requests – a few seconds delay can deter users from using the system again in the future. Another advantage of using swarms is that the “swarming” process can be easily visualised. This can be used as a way of showing users how certain items are being recommended.

Most current recommender systems suffer from a scalability problem. As the number of users increases significantly, these systems have to compromise on accuracy or speed (it is not feasible to immediately take into account changes made by a user – some systems require days or even weeks to recompute their prediction model). This thesis presents a novel swarming algorithm which, inspired by real world scenarios, allows each user to select his own group of friends (those that are considered similar to him) to give recommendations whilst avoiding those that do not share common interest and keeping in touch with those that are most similar (best friends). Indeed, this thesis presents evidence
suggesting that it is possible for a system to be adaptive, producing highly accurate recommendations, scalable to increasing number of users, and yet fast to respond.

1.2. Thesis Hypothesis

The hypothesis of this work is:

Capabilities of recommender systems can be improved by the use of evolutionary algorithms and swarm intelligence.

This thesis focuses on the following capabilities and improvements:

- **Prediction Accuracy**: given an item, a system should be able to accurately predict the user's rating for that item.
- **Relevant Recommendations**: given a user, a system should be able to provide recommendations that are considered relevant to (would potentially get a high rating from) that user.
- **User Scalability**: a system should be able to cope with increasing number of users.
- **Speed of Adaptation**: time taken for a system to learn a new user's preferences should be fast.
- **Online Practicality**: a system must be able to respond to users' requests and provide new recommendations in real-time.

1.3. Thesis Objectives

In order to provide evidence supporting this hypothesis, the following 12 objectives are defined.

(1) Identify crucial components of recommender systems from an in-depth investigation of the literature.

(2) Investigate how the use of a genetic algorithm (GA) can improve capabilities of recommender systems.

(3) Devise a method of calculating the quality of the recommendations in order that a fitness score can be assigned to the corresponding feature weights.
(4) Implement a recommender system employing a standard GA to find similar users to provide recommendations and compare its performance (in terms of prediction accuracy) with that of a non-adaptive system.

(5) Identify advantages and shortcomings of the GA recommender system.

(6) Investigate how the use of swarm intelligence (SI) can improve capabilities of recommender systems.

(7) Implement a recommender system employing an existing swarm intelligence algorithm and compare its performance (in terms of prediction accuracy and speed of adaptation) with those of the GA and non-adaptive systems.

(8) Identify advantages and shortcomings of the SI system.

(9) Build upon these advantages and shortcomings to devise a novel adaptive system.

(10) Compare the behaviour of such a system with an existing non-adaptive algorithm.

(11) Compare the performance (in terms of prediction accuracy and speed of adaptation) of the new system with those of the previous SI, GA and non-adaptive systems.

(12) Demonstrate that this system can improve capabilities (in particular, relevant recommendations) of recommender systems using a pilot study with real participants.

1.4. Contributions

In the course of this research, the following 19 contributions have been made.

(1) A representation of user profiles is devised which makes use of multiple features such as demographic information.

(2) The notion of feature weights is introduced to represent the user preferences. Once evolved or attained, they can explain how recommendations are derived and thus, increase the users' confidence in the system.

(3) A similarity measure between two users that takes into account these feature weights is proposed. This replaces a more traditional Pearson correlation that only uses a single feature, rating, as the basis for computing similarity.

(4) A standard GA is shown to be able to fine-tune a profile-matching task within a recommender system, tailoring it to the preferences of individual users.
A fitness function is devised which reformulates the problem of making recommendations into a supervised learning task, enabling fitness scores to be computed.

An experimental comparison of the performance of a non-adaptive, Pearson algorithm (PA), and GA recommender systems is made.

The importance of exploiting features other than rating is shown.

Particle swarm optimisation (PSO) is employed in a recommender system for the first time.

An experimental comparison of the performance of the PA, GA and PSO recommender systems is made.

Both adaptive techniques, GA and PSO, are shown to make more accurate predictions than the non-adaptive Pearson algorithm.

Shortcomings of the recommender system using a conventional PSO algorithm are identified and variations of PSO algorithm are proposed to overcome these limitations.

A technique to deal with multiple solutions in PSO is demonstrated.

Six distinct behaviours of PSO dynamics are identified using new analysis and visualisation tools.

Adaptation is shown to be faster for the PSO than the GA recommender system.

A novel algorithm, inspired by real-world scenarios, is described. It allows users to choose their own ‘friends’ to give recommendations. Because all users are moving in parallel, neighbourhood selection for all users can be performed simultaneously and thus, speed of adaptation is increased significantly.

The notions of ‘best friend’ and ‘outsiders’ are introduced. Best friend is used to describe the user most similar to each active user. Conversely, outsiders are those that are considered dissimilar to each user and who should be avoided. These two terms are shown to be responsible for making the system scalable and removing the need for random sampling.

A system is demonstrated where each user performs two tasks: find his neighbourhood of friends whilst attaining a set of feature weights that is best at representing his recommendation preferences.

The system has been assessed in experimental trials and received good feedback from real users.

The system architecture for an adaptive online recommender system is proposed.
1.5. Publications

The following publications are based on the work presented in this thesis:


1.6. Thesis Structure

The remainder of this thesis is structured as follows:

*Chapter 2* is divided into two main sections. Section 2.2 provides a background of all related work in the field of recommender systems. The main shortcomings of existing approaches are identified, thus highlighting the need for new techniques that can
overcome these limitations. Section 2.3 addresses two potential solutions: evolutionary algorithms (EAs) and swarm intelligence (SI). A background to these two areas is presented and shows that both EAs and SI have never been used in recommender systems.

**Chapter 3** describes a new recommender system, which employs a genetic algorithm to learn personal preferences of users and provide tailored suggestions. Section 3.1 outlines the MovieLens dataset used for all experiments (except for those in Appendix D). Section 3.2 describes the recommender system and genetic algorithm. Section 3.3 compares the performance of the GA system against that of a system using a non-adaptive Pearson algorithm. Analysis of the experimental results is presented and advantages and shortcomings of the GA system are identified.

**Chapter 4** builds on work presented in chapter 3 by employing the particle swarm optimisation (PSO) algorithm in place of the genetic algorithm to learn preferences of users. Section 4.1 compares a new PSO recommender system against the GA and PA systems. Detailed analysis of the PSO recommender is presented, identifying shortcomings in the work. Section 4.2 presents variations of the PSO recommender to overcome these shortcomings. Section 4.3 then provides extensive analysis of the final PSO recommender system using new analysis and visualisation tools.

**Chapter 5** introduces a novel algorithm, ClusterPSO, that employs dynamics similar to those found in the PSO system to mimic real-world “friend formation” scenarios. Here, users dynamically select their own neighbourhood of ‘friends’ to provide recommendations and avoid those that are dissimilar to them. Section 5.1 describes, in detail, the ClusterPSO recommender system. Section 5.2 provides preliminary cluster analysis using manual cluster allocation and an existing constrained clustering algorithm, COP_KMEANS. Section 5.3 provides experimental results and investigates the effect of altering algorithm parameter values. Analysis of the results is then presented with shortcomings identified. The chapter concludes by showing a comparison of the ClusterPSO system against all previous systems.

**Chapter 6** addresses the issues raised in chapter 5 by reintroducing the concept of feature weights to represent the user recommendation preferences. Section 6.1 describes the modified algorithm of the ClusterPSO recommender system. Section 6.2 presents the final algorithm, ClusterWeight, which builds on the modifications to the ClusterPSO...
where users now simultaneously search for both their user preferences and
neighbourhood. Section 6.3 provides experimental results and analysis of ClusterWeight.
A detailed comparison between the ClusterWeight system and all previous systems is
presented in sections 6.4 to 6.7. Section 6.8 describes a pilot study carried out with real
participants.

Chapter 7 summarises the work presented in this thesis, demonstrating that all objectives
of this research are achieved. Section 7.3 presents potential directions for future work.
CHAPTER 2

Literature Review

2.1. Introduction

Section 2.2 explores all related work in the field of recommender systems. It investigates some past and current systems and examines different approaches used in providing recommendations, with the main focus on collaborative filtering techniques which are the basis of the work presented in this thesis. The main shortcomings of these existing approaches are identified, thus highlighting the need for new techniques that can overcome these limitations. Section 2.3 addresses potential solutions, namely, evolutionary algorithms (EAs) and swarm intelligence (SI). A background to these two areas is presented and shows that both EAs and SI have never been used in recommender systems. Finally, section 2.4 concludes.

2.2. Recommender Systems

The rapid expansion of the Internet has brought about a new market for trading. Electronic commerce or e-commerce has enabled businesses to open up their products and services to a massive customer base that was once available only to the largest multinational companies. As the competition between businesses becomes increasingly fierce, customers are faced with a myriad of choices. Although this might seem to be nothing but beneficial to the customers, the sheer wealth of information relating to the various choices can be overwhelming (Schafer et al. 1999; Sarwar et al. 2000). One would normally rely upon the opinions and advice of friends or family members but unfortunately even they may have limited knowledge.

Recommender systems provide one way of circumventing this problem. As the name suggests, their task is to recommend or suggest products (in this thesis, “products” are referred to as “items”) 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 utilise knowledge of customers' likes and dislikes to build an understanding of their individual needs and thereby increase customer loyalty (Schafer et al. 1999; Sarwar et al. 2000; Schafer et al. 2001). This is known commercially as B2C (Business-to-Consumer) relations and applies to businesses dealing with large number of customers over the internet (Novo 2002).

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 (Breese et al. 1998; Goldberg et al. 2000). Conversely, others regard "recommender system" as a generic descriptor that represents various recommendation/prediction techniques including collaborative filtering, content based filtering, Bayesian networks and association rules (Resnick and Varian 1997; Terveen et al. 2001; Delgado 2000). In this research, we adopt the latter definition when referring to recommender systems.

Malone et al. (1987) proposed three types of information filtering techniques: cognitive, social and economic. A cognitive approach is based on analysing the contents of an item or document. A social technique works by employing the personal and organisational interrelationships of individuals in a community. An economic approach relies on estimated implementation and search cost and benefits of use. This thesis is focused on collaborative filtering techniques which are based on Malone’s second approach of social filtering. This is chosen as it seems to be the form that is closest to the process of providing recommendations in real-life i.e. from friends or family members.

2.2.1. Collaborative Filtering

A simple recommender system can be implemented using database queries, whereby the mean (average) rating by existing users for an item is used as the predicted rating for that item for new users. However, this simple approach does not produce personalised recommendations. Thus, most commercial recommender systems use a collaborative filtering technique to generate recommendations. Collaborative filtering was proposed by Shardanand and Maes (1995) for automating the process of "word of mouth" recommendations. The idea behind it is that people with similar tastes should be able to recommend items to each other. A simple collaborative filtering system starts by building up profiles of each user and then using an algorithm such as nearest neighbour (each new
user is compared with existing users using a distance metric, and the closest existing users are considered most similar to the new user) to find profiles similar to that of the current user. Note that this thesis uses the word “user” to mean “user profile” throughout. The current user is referred to as the *active user*, A. Selected data from those profiles are then used to compute predicted ratings for unseen items (those with no actual rating from A) and thus, build recommendations for A.

Breese et al. (1998) proposed that the predicted rating for the active user A on item i, denoted *predicted_rating(A, i)*, is a weighted sum of the ratings of the other users for i.

*predicted_rating(A, i)* is defined as:

\[
\text{predicted_rating}(A, i) = \text{mean}_A + k \sum_{j=1}^{n} w(A, j)(\text{rating}(j, i) - \text{mean}_j) \quad [1]
\]

where:

- \( \text{mean}_j \) is the mean rating for user j, thus

\[
\text{mean}_j = \frac{1}{m} \sum_{j=1}^{m} \text{rating}(j, i)
\]

- \( \text{rating}(j, i) \) is the actual rating that user j has given on item i, if it exists.
- \( k \) is a normalising factor such that the sum of the weights is equal to 1, thus

\[
k = \frac{1}{\sum_{j=1}^{n} w(A, j)}
\]

- \( n \) is the number of users.
- \( m \) is the number of items that user j has rated.

The weights, \( w(A, j) \), reflect distance or similarity between the active user A and a user j. The most commonly used method proposed by Resnick et al. (1994) to measure similarity is the Pearson correlation coefficient, *correlation(A, j)*, which can be defined as:

\[
\text{correlation}(A, j) = \frac{\sum_{i=\lambda_1}^{\lambda_z} (\text{rating}(A, i) - \text{mean}_A)(\text{rating}(j, i) - \text{mean}_j)}{\sqrt{\sum_{i=\lambda_1}^{\lambda_z} (\text{rating}(A, i) - \text{mean}_A)^2 (\text{rating}(j, i) - \text{mean}_j)^2}} \quad [2]
\]

where:

- \( \text{mean}_j \) is the mean rating for user j.
- \( \text{rating}(j, i) \) is the actual rating that user j has given on item i.
- The common items that users A and j have rated are defined as the set \( \lambda_1 \ldots \lambda_z \).
- \( z \) is the number of common items.
It is obvious that user profiles play a vital role in most collaborative filtering systems. In his thesis, Mastenbroek (1999) lists three possible ways to construct and maintain a user profile. The first method, called explicit feedback, involves the user participating e.g. giving scores to certain items. The second, implicit feedback, is used when the system automatically infers the user's preferences from his actions. When explicit feedback is used, a relatively accurate profile is created from the user's response. By contrast, implicit feedback reduces the amount of direct user involvement but it also introduces a greater margin of error. However, in order to benefit from the advantages of both techniques, a hybrid system is then required. Implicit feedback can be used to capture information about the user on his first visit, which will then generate an initial user profile. This profile can then be modified and enhanced using explicit feedback from the user. In this case, explicit feedback is used as a correction mechanism.

Based on these methods of maintaining a user profile and the level of user participation required, collaborative filtering systems can be grouped into three categories: *active*, *passive* or *automated* (Chislenko 1997; Herlocker et al. 2000). In this thesis, these are defined as follows:

- **Active Collaborative Filtering** – the users are required to specify their preferences by subscribing to a community. The system then presents information or items that are of interest to each user when experts in his/her community recommend them.
- **Passive Collaborative Filtering** – the users are required to explicitly rate items. The system then gives recommendations based on these ratings.
- **Automated Collaborative Filtering** – instead of asking users to explicitly rate items, the system can use methods to infer user preferences without direct participation from the users.

These categories are described in more detail below, giving examples of earlier systems that fall under each category. However, it is worth mentioning that in recent years, many systems have been implemented using more complicated techniques and their features may overlap between categories. Thus, these categories are beginning to merge into a single category, simply known as automated collaborative filtering (Chislenko 1997).
Active Collaborative Filtering

Active collaborative filtering is commonly found in earlier work in this field and takes on many forms. One of these forms involves querying people for information. This is useful when a user needs to find particular information by 'pulling' knowledge out of the community. An example of this would be search engines. The drawback of using this 'user pull' method to find items is that the results are relevant and useful only if the user knows exactly what she is looking for and what sort of question she needs to ask. Because of this drawback, most active collaborative filtering systems adopt the 'user push' method where the user subscribes to a community and shares her requirements and/or preferences. Members of the community then take an active role in supplying the user with important knowledge i.e. recommendations and suggestions. In this approach, the information is 'pushed' towards the user by the community (Delgado 2000). The success of this method relies on the participation of the members of the community sharing their knowledge with others.

Tapestry, an experimental mail system developed at Xerox PARC, was the first system to support collaborative filtering (Goldberg et al. 1992). It allows the users to annotate the documents they read and specify whether they “like it” or “hate it”. These annotations are shared by the users in the system, allowing them to filter their incoming documents with greater ease and focus on items that are of more interest to them.

The GroupLens\(^2\) research group at the University of Minnesota has been working in the fields of information filtering, collaborative filtering and recommender systems since 1992 and has published a vast amount of literature in these areas (Resnick et al. 1994; Miller et al. 1997; Konstan et al. 1997). Their early project, collaborative filtering of Netnews (Resnick et al. 1994), falls under the category of active collaborative filtering. In 1996, the PHOAKS (People Helping One Another Know Stuff) system was implemented which automatically reads, classifies and extracts recommendations from messages in Usenet newsgroups. The difference between PHOAKS and a search engine is that rather than presenting all documents that match a specified keyword, PHOAKS offers only a list of relevant documents that are recommended by a community of experts. The documents are ordered by frequency of being mentioned, and opinions about the documents are also presented (Hill and Terveen 1996; Terveen et al. 1997). Explicit user ratings are not

\(^2\) [http://www.grouplens.org/](http://www.grouplens.org/)

35
required; however, user participation (in the form of users posting messages to the website) is needed as the system relies on them as the source of recommendations.

A system by Maltz and Ehrlrich (1995) relies on people who find interesting documents actively sending pointers of those documents to their colleagues. A pointer contains a hypertext link to the source document as well as contextual information to help the recipient determine the interest and relevance of the document prior to accessing it.

Dooyoo\(^3\) 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 to the website will often submit reviews of 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! The service is similar to that found at Amazon.com\(^4\), located on the item information page, where recommendations based on the opinions of other customers are presented in the forms of written reviews and 1-5 star ratings.

Other work based on active collaborative filtering includes ReferralWeb (Kautz et al. 1997) which uses the idea that there are six degrees of separation between all the people in the world – any two individuals of the entire human population are likely to be connected through a short sequence of acquaintances. The system mines public data sources to find the person the user wants to meet. Its shortcoming is that the user is required to explicitly specify either the name of the person he/she is searching for or the topic on which an expert is needed. As with most active collaborative filtering systems, ReferralWeb relies heavily on user participation.

**Passive Collaborative Filtering**

Ringo: a personalised music recommendation system (Shardanand and Maes 1995) was one of the earlier systems that made use of passive collaborative filtering. Note that this was a prototype that led to the spin-off company, Firefly that was later bought by Microsoft. The system maintained a profile for each user - a concept that was central to the way it worked. A profile was a record of a user's interest in various items. Each user

\(^3\) [http://www.dooyoo.co.uk](http://www.dooyoo.co.uk)
\(^4\) [http://www.amazon.com](http://www.amazon.com)
would describe his music preferences by rating some music. The user's profile would change over time as the user rated more music. The system worked by comparing profiles to determine which ones were similar. Once a set of very similar profiles had been identified, the system was then able to predict the opinion the user would have on an artist or album that had not yet been rated. This was achieved by computing a weighted average of all the ratings given to that item by other users who had similar profiles.

Passive collaborative filtering is the most commonly used technique to obtain user preferences. For more examples, see Related Work on Movie Recommender Systems later in this chapter.

**Automated Collaborative Filtering**

Active and passive collaborative filtering techniques only work well if people play an active role in evaluating or giving ratings to items. The problem arises when people do not want to spend time doing such things. To avoid this, automated collaborative filtering systems have been implemented to work in the same way as passive collaborative filtering, but they can create user profiles without direct participation from the user. An automated collaborative filtering technique uses *observational ratings* that infer a user's preferences from his actions rather than requiring him to explicitly rate an item (Nichols 1997; Mastenbroek 1999). These observational ratings can include browsing data, information access patterns or purchase history from online stores (Breese et al. 1998; Terveen et al. 2002). Work by Morita and Shinoda (1994) shows that there is a relationship between time spent on items and explicit ratings by the users. Results from the experiments with Usenet news, carried out by Konstan et al. (1997), also confirm this finding that predictions based on time spent reading articles are nearly as accurate as predictions based on explicit numerical ratings.

One of the biggest e-commerce websites, Amazon.com, introduced a recommendation service called BookMatcher which used traditional collaborative filtering to give recommendations on books based on the ratings of customers with similar tastes. By this method, this recommender algorithm would have fallen under the above category of passive collaborative filtering. However, this service has since moved towards the category of automated collaborative filtering as the current service does not require customers to explicitly rate items (Wolverton 2000). Although customers can still choose
to rate items, the website now uses their viewed and purchased items to compute recommendations. The algorithm, item-to-item collaborative filtering, works by matching each user's purchased and rated items to similar items. A list of most similar items is then presented to that user as recommendations (Linden et al. 2001). According to Linden et al. (2003), their amazon.com recommendation algorithm outperformed traditional collaborative filtering methods on scalability, cluster models on recommendation quality and search-based models on both scalability and recommendation quality. Because the algorithm only considers the items that have been either bought or rated by the customers and does not take into account other user information such as age, gender or occupation, the recommendations will often be similar to the original items but fail to introduce new items from different categories.

Other systems under the category of automated collaborative filtering include Beehive which uses the behaviour and interactions of members to automatically update user profiles (Huberman and Kaminsky 1996) and Siteseer which uses personal bookmark lists to infer user preference (Rucker and Polanco 1997).

Related Work on Movie Recommender Systems

The work in this thesis focuses on movie recommendation so this section outlines work in this area.

Several systems that recommend movies have been developed and studied in recent years. Most systems are based on the same concept and in order to distinguish between them, the way in which a group of similar users is chosen and how predicted ratings are calculated must be examined (Breese et al. 1998).

Bellcore's Video Recommender, developed by Hill et al. (1995), is an email interface recommender system for movies. Initially, a user sends an email to join and the system replies with a list of 500 movies for the user to rate on a scale of 1-10. A group of most similar users are then found and used to give recommendations. The system also allows joint recommendations for more than one person so for instance, users A and B can ask to be recommended movies to watch together. One disadvantage of Bellcore's recommender is that for every new user, a prediction equation (responsible for computing predictions for future items) has to be produced which can take some time as the system has to look
for correlations between the active user and other users in the database. Thus, the initial recommendations can be slow. Furthermore, any changes in a user’s preferences require his prediction equation to be updated and can also cause other prediction equations which depend on this user to be recomputed. This could take a considerable amount of time if the number of users is large.

*MORSE* at BTExact (Fisk 1996) was deployed in 1996 with two requirements in mind: *accessibility* by as many people as possible, and *ease of use*. To achieve these requirements, MORSE was put on the World Wide Web and used forms to obtain ratings. In MORSE, the predicted rating for the active user on item *i* is estimated by normalising the ratings of other users on the same item. Each recommendation is presented to the users with an estimated error on it to help them decide whether to trust the provided recommendation. Compared to Bellocore’s video recommender, MORSE provides a more interactive experience for the users, allowing them to obtain information about the films. In addition to personalised recommendations, the users can view the average ratings for chart movies. However, because MORSE also requires the active user to be compared against other users in the database, the problem of slow initial response time (the time taken to calculate predictions) still exists.

Other movie recommendation work includes Compaq Systems Research Centre’s *EachMovie* (McJones and Detreville 1997), *MovieFinder* and a system by Cayzer and Aickelin (2002) based on the Immune Network (described later in the Immune Systems section). The most successful system, however, is *MovieLens* (Herlocker et al. 2000) from the GroupLens research group. The system again works on the same principle of finding similar users that provide recommendations to each other. To overcome the problems of slow response time and changes in user preferences found in conventional collaborative filtering systems, an item-based approach is used to replace the search of similar users. Recommendations are computed using similar items to items that the active user has liked. It is assumed that relationships between items are relatively static and therefore, the item neighbourhoods according to their similarity values can be precomputed which results in faster response time. However, a drawback to most item-based algorithms is that because recommended items tend to be similar to those rated by the active user, the chance of serendipitous discovery is reduced. Various studies and

---

3 http://www.moviefinder.com
4 http://www.movielens.org
experiments have been carried out in recent years to improve quality of recommendations provided by MovieLens which led to many features such as group recommendations where recommendations are produced for groups of users rather than for individuals (O'Connor et al. 2001), confidence display showing a level of confidence in each movie recommendation (McNee et al. 2003) and MovieLens Unplugged, allowing MovieLens to run on a PDA (Miller et al. 2003) being added to the system. (The dataset collected through the MovieLens website has been made available for research purposes and is used for experiments in this thesis).

All of the above fall under the category of passive collaborative filtering where users are asked to rate movie items. However, the work by Mukherjee et al. (2001) with Movie2Go differs from the others in that it does not make use of collaborative filtering, instead it incorporates a voting based ranking technique with a Bayesian learning mechanism (Mitchell 1997) to capture the frequency of commonly occurring keywords in the synopsis of movies that a user rates. This is used to deduce user preferences of various features such as most/least favourite actors, movie genres and directors; these can be explicitly modified by the user. A drawback to this system (or any learning method) is that in order for it to successfully capture the preferences for all the features automatically, the users themselves are required to have rated some movies across every feature type.

Evaluation Metrics

Since this thesis focuses on different approaches to improve recommender system prediction accuracy, it is important to be able to compare and evaluate the performance of such systems. Herlocker et al. (1999) define three key areas on which the quality of a prediction algorithm can be measured:

• **Statistical Accuracy** – Statistical accuracy metrics evaluate the accuracy of a collaborative filtering system by comparing the predicted ratings against the users’ actual numerical ratings. Many metrics have been proposed for assessing this type of accuracy such as Mean Absolute Error (MAE) (Shardanand and Maes 1995; Breese et al. 1998; Sarwar et al. 2001), Root Mean Squared Error (Sarwar et al. 1998) and correlation between actual ratings and predictions (Hill et al. 1995; Konstan et al. 1997). However, the MAE is the most widely used
metric that measures the deviation of recommendations from the users’ actual ratings. MAE can be defined as:

\[
\text{MAE} = \frac{1}{N} \sum_{i=1}^{N} | \text{predicted}_{rating}(A,i) - \text{rating}(A,i) | 
\]

[3]

The lower the MAE the more accurate the system is at predicting ratings. Herlocker et al. (1999) performed an empirical analysis on different statistical accuracy metrics and found that they generally provide similar conclusions.

- **Decision Support Accuracy** - Decision Support Accuracy metrics evaluate the effectiveness of a system in helping a user select high quality items from the item set. They assume a binary assessment; the user thinks the recommended item is either **good** or **bad**. With this type of accuracy, a threshold of ‘good’ items has to be specifically defined. The most commonly used metric is the **Receiver Operating Characteristic** (ROC) (Herlocker et al. 1999; Sarwar et al. 2001). The accuracy is measured by the area under the ROC curve that plots the **sensitivity** (on the y-axis) and **specificity** (on the x-axis) of the system. Sensitivity refers to the probability of a randomly selected good item being accepted by the user. Specificity, on the other hand, refers to the probability of a randomly selected bad item being rejected by the user. A threshold value is usually set to 4 where the ratings of 4 or above are considered good recommendations by the user. The range of ROC is 0 to 1 where 1 is perfect (the system is able to predict with 100% accuracy), 0.5 is random and 0 is imperfect (the system always makes wrong predictions).

Sarwar et al. (1998) reported that from their experiments, MAE and ROC provide the same ordering of different experimental schemes in terms of prediction quality.

- **Coverage** – coverage measures the percentage of items for which a collaborative filtering system can provide predictions. Coverage can reduce if there are no ratings available for certain items or possibly when there are few ratings available but they belong to the users that have zero correlations with the active user. The prediction formula used to compute predicted ratings in this thesis provides 100%
The active user’s mean rating is used as the predicted rating for those items with no available ratings.

The evaluation metric used in this thesis is slightly different to those described above due to the nature of the results obtained from using stochastic algorithms (particle swarm optimisation and genetic algorithm). A new evaluation metric (explained in detail in chapter 3) is devised to evaluate the results by considering only the percentage of predictions which are correct (determined by either zero or at-most-one tolerance level). In the zero tolerance level, a prediction is considered correct if the predicted and actual ratings given by the active user are exactly the same. Similarly, the at-most-one tolerance level considers a prediction to be correct if the difference between the predicted and actual ratings is at most 1.

**Shortcomings of Collaborative Filtering Systems**

There are four most commonly experienced problems with traditional collaborative filtering that have been identified (Lee 2001). These are listed as follows:

- **New User Problem**: When a new user joins the system, he/she expects to get accurate predictions or recommendations with little delay. If the system fails to do so then the user may lose patience and abandon the system.
- **Recurring Startup Problem**: New items are added to recommender systems regularly. However, the system may not be able to compute an accurate prediction for them, had they not yet been rated by any users.
- **Sparse Rating Problem**: This problem is very common. The number of ratings obtained can be low, especially during the initial phase. Thus, there are few overlaps of ratings between users.
- **Scaling Problem**: Most recommender systems involve a large number of users. As the website grows, the computation complexity needs to scale well with increasing number of users and items.

Most work on recommender systems deal with the first type of problem: to improve the accuracy of predictions and increase speed in which these predictions are computed (Breese et al. 1998; Good et al. 1999; Kohrs and Merialdo 1999). However, from my
experience with existing systems, the problem with the prediction accuracy and relevance of recommendations still exists.

The second problem of unrated items can be solved by using content-based methods that make use of the contents of items rather than ratings to compute predictions. However, a trade-off for using this approach is that some content-based algorithms can increase response time (Lee 2001). Perhaps, a simpler solution would be to use the average rating of the user as the predicted rating for new items.

Various models have been proposed to solve the sparsity problem. Billsus and Pazzani use the singular value decomposition (Berry et al. 1995) as a dimensionality reduction technique to project user ratings and rated items into a lower dimensional space. This allows users to make recommendations to each other even without any overlap of rated items (Billsus and Pazzani 1998). The sparsity problem can also be solved using an algorithm employed in Alkindi (Stern 2001) (described later in clustering section). In addition, Breese et al. (1998) presented the idea of default voting which is an extension to the Pearson correlation algorithm (Resnick et al. 1994) used in many existing recommender systems to calculate correlation or similarity between two users. Instead of using only common movies that both users have rated, all movies that either both or one of them have rated are employed. A user-defined default value is assumed for items where an explicit rating from one of the users does not exist. This default voting approach was able to produce satisfactory results (Breese et al. 1998). This idea can also be extended to tackle the previous recurring startup problem where the default value is used for a number of additional items that neither of the two users has rated.

Goldberg et al. (2000) proposed an algorithm called Eigentaste which attempts to overcome all four problems. It differs to most CF systems in that their computation complexity does not scale linearly with the number of users, \( n \), in the system, \( O(n) \). Instead, it claims to provide recommendations to users in constant online time, \( O(1) \), hence a fast response time. Moreover, each user in Eigentaste is initially presented with universal queries i.e. the same set of items to rate to avoid the sparsity problem (Sarwar et al. 1998) which is normally the case if the user is allowed to pick items to rate. The algorithm is split into two computation tasks, online and offline. The offline phase uses principle component analysis, introduced by Pearson (1901), for optimal dimensionality reduction. Users are then clustered in the lower dimensional subspace. The online phase
uses initial ratings in the form of eigenvectors to project new users into clusters. A lookup table is then used to present users with recommendations, thus the runtime is independent of the number of users in the system. Jester, an online joke system, was implemented to evaluate Eigentaste and compare its performance against other algorithms such as Popularity (POP) where the most popular items are used as recommendations, and the nearest neighbour algorithms where a set of most similar users is used to provide recommendations (Breese et al. 1998). Even though the online computation time for Eigentaste is very fast, a drawback is that the system is not adaptive to changes in user profiles. Reclustering users can take up a long time especially if the number of users in the system is large, thus this process cannot be done every time a change occurs.

In addition to the four problems described above, a shortcoming that most websites using collaborative filtering suffer from is that they do not have any facility to provide explanations of how recommendations are derived. This was addressed by Herlocker et al. (2000) who proposed explanation facilities for recommender systems in order to increase users' faith in the suggestions. Various efforts have also been made to increase trust in recommendation algorithms. Swearingen and Sinha (2002) suggest the keys to increase trust in a recommender system are:

- To make the functionality underlying the algorithm transparent to the users.
- To provide details about the recommended items such as pictures and community ratings.
- To allow the users to refine recommendations they receive by including or excluding items.

For this reason, McNee et al. (2003) have added a basic confidence display to the MovieLens recommender system and shown that user satisfaction has increased. Moreover, a study by Cosley et al. (2003) has also shown that recommender interfaces do affect users' opinions, in particular, the way in which they give ratings to items when the rating scale and the display of predictions are shown.

### 2.2.2. Alternative Techniques

There are many techniques which have been used in order to address the issues found in collaborative filtering methods described above and hence, further improve performance
of recommender systems. The following methods are most relevant to the area of recommender systems.

**Content-based Filtering**

A content-based or cognitive filtering approach referred to by Malone et al. (1987) differs from collaborative filtering in that it analyses the contents of the items being recommended by comparing representations of content contained in the documents to representations of content that the user is interested in (Herlocker et al. 1999). This method has shown to be effective in recommending textual documents (Maes 1994; Basu et al. 1998; Mooney and Roy 2000).

An example of this is LIBRA: the Learning Intelligent Book Recommending Agent, which combines a content-based approach with machine learning, namely Bayesian text classifier (Mitchell 1997) to make book recommendations. Each user is treated individually - there is no sense of "community" which forms the basis of collaborative filtering. It has an advantage of being able to recommend unrated items to users with unique interests and to provide explanations for its recommendations (Mooney and Roy 2000). There are, however, a few limitations with the content-based approach:

- The content-based filtering does not work with items whose contents are not easily analysed or described such as images and songs.
- It cannot base recommendations on quality or taste.
- Recommendations produced are usually what the user has previously seen (or indicated liking); most content-based filtering systems have no inherent method for serendipitous discovery.

It has been shown that, in general, collaborative filtering systems can achieve higher prediction accuracy than those employing content-based algorithms (Melville et al. 2001). However, in some cases, a collaborative filtering system is unable to provide predictions for certain items whose ratings are not available. In such cases, a content-based algorithm can be used as it does not rely on user ratings to provide recommendations. Mooney (Melville et al. 2001; Melville et al. 2002), one of the researchers behind LIBRA, has since moved onto integrating features from both content-based and collaborative filtering

---

approaches to form a hybrid recommender system that overcomes the shortcomings found in pure content-based systems, namely sparsity and new user problems, and provides more personalised recommendations through collaborative filtering. Similarly, Good et al. present a hybrid system that combines personal information filtering agents which analyse item content and develop a personal user interest profile with collaborative filtering to produce better recommendations (Good et al. 1999). One common problem found with most hybrid systems using collaborative filtering and content-based filtering is that the accuracy of a content-based filtering algorithm depends on the number of movie items the user has rated. In the event that the collaborative filtering part cannot produce predictions due to unavailable ratings, the system can only benefit from the content-based part if the number of rated movie items is high.

Similar work on combining content-based and collaborative filtering systems includes FAB, a system developed by Balabanovic and Shoham (1997) at Stanford, that presents a user with web pages that may be of interest. The users are often required to comment on the websites that they visit, hence too much burden is placed on the user which can discourage some users in actively using the system. Only when a lot of other people have made this time investment does the information become useful.

**Classification**

Another approach that we can adopt is knowledge discovery through the use of data mining. Here, a recommender system will be a large multi-user application that is continually gathering various data, such as new rules, inventory and user preferences.

Classification techniques are an example of supervised learning. It essentially has two phases – learning and prediction. Initially a dataset (training data) is used to train the algorithm, the aim of which is to formulate a decision tree model or a set of “if-then” rules. Once a model has been obtained, it can be used to predict the class of new unclassified data (Mitchell 1997).

In the context of recommender systems, classification can be applied to mine all past transactions of all users. The resulting model allows us to predict whether or not a given user will like a certain product. One of the famous decision tree classification techniques is C4.5, introduced by Quinlan (1993). It is well-known for its speed and accuracy and is...
often used as a benchmark for other algorithms (Ujjin 1999). Other work includes a system developed by Billsus and Pazzani (1998) where a neural network was trained to perform a classification.

Clustering

As the final systems in chapters 5 and 6 of this thesis use the idea of clustering to group users that are similar, this section examines and describes various clustering techniques in more detail.

Clustering is one form of unsupervised learning and considered a tool of discovery because it has the potential to reveal previously undetected relationships based on complex data. Unlike classification, there is little or no input from the environment and there are no learning and prediction phases. However, by using supervised learning, clustering can also be used as a classification tool whose output is generated in the form of clusters, rather than rules or trees. In supervised clustering, each data item has an associated output class value. Although, this output field is not used in the clustering, it is used in devising a suitable metric defined on other fields such that data items with the same class value are grouped together (Al-Harbi and Rayward-Smith 2003). Supervised clustering has been applied to domains such as document classification (Aggarwal et al. 1999), breast cancer and diabetes data (Al-Harbi and Rayward-Smith 2003) and cancerous genes (Dettling and Bühlmann 2002). With current data used in this thesis, users do not have a predefined class in which he/she belongs, thus only unsupervised clustering can be used to group similar users. For this reason, whenever clustering is mentioned in this thesis, it refers to unsupervised clustering.

A simple clustering approach attempts to identify similar data items and group them according to the degree of association between the items and groups i.e. the dataset is divided into groups of homogenous sub-groups or clusters. Various clustering analysis techniques exist and can be categorised into two main types: non-hierarchical and hierarchical. A non-hierarchical approach divides the dataset of \( n \) items into \( m \) clusters. An example of this is the \( k \)-means clustering algorithm where \( k \) defines the required number of clusters (MacQueen 1967; Hartigan 1975; Hartigan and Wong 1979), which is by far the most popular clustering method. A basic \( k \)-means algorithm starts by randomly selecting \( k \) points as cluster centres. Data items are then assigned to their closest cluster
centre measured by a distance metric (the most commonly used is the Euclidean distance function). The mean of all data items in each cluster is then computed and used as new cluster centre for that cluster. The process is repeated until the same data items are assigned to each cluster for consecutive rounds. Alternatively, a hierarchical method produces a nested dataset in which pairs of items or clusters are successively linked. This hierarchical approach can be categorised into 2 basic types: agglomerative (bottom-up) and divisive (top-down) (Jain and Dubes 1988; Kaufman and Rousseeuw 1990). An agglomerative approach starts with clusters, each containing a single data item (singleton), and then recursively grouping together two or more most similar clusters until the entire dataset is encapsulated into one cluster. Conversely, a divisive technique begins with a single cluster and then recursively splits the dataset into smaller clusters until each cluster consists of only a single data item. Methods for measuring distances between clusters include single linkage, complete linkage, centroid and median (Gose et al. 1996; Timmis 2001a). The results of hierarchical clustering are usually represented as a tree or dendrogram.

Most real world problems are usually subjected to some problem-specific limitations (Berkhin 2002) and that the same dataset may need to be partitioned differently for different purposes. An example given in (Jain et al. 1999) is that of a whale, an elephant and a tuna fish. Partitioning them based on ‘mammals’ would result in the whale and the elephant being in the same cluster. However, if the constraint is now changed to ‘living in water’, the whale would then need to form a cluster with the tuna fish instead. Other constraints include parameter constraints or constraints on individual clusters involving bounds on aggregate functions such as minimum, average and maximum over each cluster (Berkhin 2002). A frequently occurred constraint, however, is the minimum number of data items required in a cluster. Unfortunately, the traditional k-means algorithm sometimes results in very small clusters (or empty clusters). Bradley et al. (2000) suggest a modification of the k-means algorithm by imposing a minimum size of 1 on each cluster. This requires that if a cluster becomes empty (due to the data item which acted as the previous cluster centre moving to join a different cluster), the step is then reverted and the previous cluster centre is kept for this cluster. This then ensures that there is at least one data item in each cluster. Work by Neal (2002) (described later in Immune Systems section) automatically deals with this constraint. Data items that do not fit into any clusters are removed from the system due to decreasing stimulation level over time. Interestingly, Tung et al. (2001) presented a clustering algorithm to overcome the
problem that data items could be obscured by physical obstacles such as rivers, lakes and highways. To solve this problem, Euclidean distance, most commonly used to measure similarity between 2 items, is replaced by obstacle distance which is the length of the shortest path between 2 points. Wagstaff et al. (2001) proposed COP-KMEANS, a constrained \(k\)-means clustering algorithm, which incorporates background knowledge that can be expressed as a set of instance-based constraints. These constraints state which instances (data items) should or should not be grouped together. In their previous work, Wagstaff and Cardie (2000) introduced two types of pair-wise constraints, must-link and cannot-link. Must-link constraints specify pairs of data items which must be placed in the same cluster. Conversely, Cannot-link constraints specify pairs of data items which must not be assigned to the same cluster. Although only must-link constraints are transitive, checks must be performed on both kinds of constraints. For example, if data item \(d_i\) must link to \(d_j\) and \(d_j\) cannot link to \(d_k\) then \(d_i\) also cannot link to \(d_k\). Although COP-KMEANS is relatively simple to implement and has shown to work successfully well (Wagstaff et al. 2001), there are a few limitations:

- the constraints must be specified as a pair of data items that are either ‘must-link’ or ‘cannot-link’.
- some data items can fail to be assigned to any cluster (due to constraint violations).
- the algorithm is not adaptable to changes in the constraints. Any changes made to the set of constraints can make the existing clusters become inaccurate and would require repartitioning.

The full algorithm of COP-KMEANS and its implementation are described later in chapter 5.

In the context of recommender systems, clustering techniques can be used to identify groups of users who appear to have similar preferences (Sarwar et al. 2001). Once the clusters are created, the predictions can be made using the opinions of the users in that cluster. Clustering methods can produce less-personal recommendations than other techniques, and work by Breese et al. showed that in some cases, the accuracy obtained from the clusters is worse than those obtained using the Pearson correlation algorithm to find nearest neighbours (Breese et al. 1998). Nevertheless, clustering can be applied as a first step to reducing the computation load of other methods such as nearest neighbour algorithms by reducing the size of the group to be analysed (Schafer et al. 2001; Berthold and Hand 2003). By combining clustering and classification, it is possible to generate a
highly accurate classifier that describes the characteristics of the different clusters that have been identified. Then when a new user registers on the system we can predict, with reasonable accuracy, the group to which he or she belongs.

O'Connor and Herlocker (2000) carried out experiments which use four existing clustering algorithms: average link hierarchical agglomerative (Gose et al. 1996), ROCK (Guha et al. 1999), $k$Metis and $h$Metis (Karypis 2002), to partition the set of items based on user rating data. Predictions are then computed independently within each partition. This therefore increases scalability in terms of the number of items in the system and decreases computation time as there is less data to consider. The Pearson correlation coefficient is used to calculate a similarity between items.

Alkindi movie recommendation engine (Stern 2001) uses a slightly different approach to provide recommendations. It uses two types of clusters: movie and user. Initially, all movies are partitioned based on genres resulting in $n$ movie clusters where $n$ is the number of genres. All movie items in each movie cluster that have received at least a given number of ratings are considered core products for that cluster. Additionally, users who have rated at least one core product in a movie cluster are assigned to that cluster. It is therefore common for a user to belong to several movie clusters. The system considers each movie cluster separately. A standard $k$-mean algorithm is then employed to partition users in each movie cluster into $k$ user clusters based on their ratings of items that are core products. Recommendations are then made to each active user by presenting movies that receive a high average rating in that movie cluster. For this reason, if the user does not rate any movies belonging to a certain movie cluster; he will not receive any recommendations from that category. Although the system performs well with sparse data, an obvious drawback is that movies from genres which the user has not rated cannot be recommended. This clustering process is performed at regular intervals. If new ratings are received in between these intervals, the system attempts to assign the user to a more appropriate user cluster. This also applies to a new user where the system finds the closest user cluster for each movie cluster that the user belongs to. However, these new ratings either from existing or new users do not contribute to the recommendations given to other users. These changes are only taken into account when the whole set of users is reclustered. Thus, the accuracy of the cluster model depends on the frequency of the update.
Other works which proposed clustering algorithms based on the Immune Network metaphor (Timmis 2001a; Hart 2002; Neal 2002) are described later in the Immune Systems section.

Bayesian Networks

Bayesian methods provide the basis for probabilistic learning methods that accommodate or require knowledge about the prior probabilities of alternative hypotheses and about the probabilities of observing various data given the hypothesis (Mitchell 1997).

Bayesian Networks are used to construct a graphical model that encodes probabilistic relationships among all variables of interest (Heckerman 1996). A Bayesian network describes the probability distribution governing a set of variables by specifying a set of conditional independent assumptions along with a set of conditional probabilities, making it a convenient way to represent causal knowledge.

Breese et al. (1998) employed a Bayesian network to produce a model based on training data. In the resulting model, each node corresponds to each movie item in the system and has a set of parent items that are used to predict its rating. A conditional probability table is constructed for each item/node, describing the probability distribution for that item given the values of its immediate predecessors. This is usually represented by a decision tree. Figure 2.1 shows an example of a decision tree for television viewing data adapted from (Breese et al. 1998). This approach has shown to be as accurate as nearest neighbour methods where a set of most similar users is used to provide recommendations (Breese et al. 1998). However, the model can be built off-line and take a long time. Thus, it is not suitable for domains where user preferences change constantly, requiring the model to be updated frequently (Schafer et al. 2001; Sarwar et al. 2001).
Association Rules

Association rules associate a particular conclusion (e.g. the purchase of a particular product) with a set of conditions, such as the purchase of several other products (Agrawal and Srikant 1994; Houtsma and Swami 1995; Hipp et al. 2000). In other words, association rules find correlations between various items in the database.

An example of association rules in use is the Amazon website. One of their well-known recommendation techniques is “people who bought this book also bought...”. Similar methods can be used, such as suggesting to customers who bought a shirt that they might want to consider buying a tie as well.

Immune Systems

Another approach that has been used in the recommender systems field is artificial immune systems (AIS). AIS are defined as adaptive systems that are inspired by the immune system and applied to problem solving (Timmis 2001a; Hart 2002; de Castro and Timmis 2002).
Cayzer and Aickelin (2002) presented a movie recommender system that is based on the immune network. An artificial immune system is employed 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 algorithm allows the concentrations of antibodies that provide a better match to the antigen to increase and eventually only a subset of good matches would remain in the system. The goal is to use the resulting set of antibodies that are a close match but at the same time distinct from each other to provide recommendations.

Timmis (2001a) proposed the first AIS for data analysis whose aim is to generate a meaningful representation of the training data in the form of a network. The resulting network is represented by B-cell objects and links between those objects, see figure 2.2 below for a diagram taken from (Timmis 2001b). Each B-cell object contains a data item, a matching mechanism and a record of the stimulation level for that B-cell. The network adapts over time with similar B-cells linking together and results in clusters of similar B-cells. This idea could perhaps be employed in a recommender system to cluster similar users where each B-cell represents a user profile.

![Figure 2.2: A network produced by AIS, taken from work by Timmis (2001b)](image)

Various enhancements to Timmis’s system were implemented (Timmis 2001a) and led to the creation of the Self-Stabilising Artificial Immune System (SAIS) by Neal (2002; 2003) which is stable, adaptive and allows for continuous learning. Although, SAIS exhibit adaptivity property (which can be used in recommender systems to adapt to changes in user preferences over time), SAIS uses a dynamic clustering algorithm which is deterministic where consistent output networks are produced given the same dataset. To achieve the deterministic property, the system requires each data item to be compared to
every existing node in the network; this involves high computational overheads and could take a long time which is not ideal in real-time recommender systems with thousands of users.

### 2.2.3. Summary of Recommender Systems

To summarise, much work has been done in the area of recommender systems. By examining different techniques used to provide recommendations, three main problems have been identified:

- **Slow response time.** Recommender systems usually run online and therefore a fast response time is crucial. Most techniques require a prediction model (a set of similar users) to be computed for each new user. This can take considerable time to produce, making this inappropriate for real-time systems.

- **Not adaptive.** Because user preferences change over time (adding more ratings, modifying or deleting existing records), the system needs to be able to adapt to these changes in order to improve the accuracy of their recommendations. Most systems require their prediction model to be updated or recomputed offline periodically. Between these updates, if there are major changes in the user preferences then the existing model can become inaccurate, resulting in poor recommendations being produced.

- **Does not scale well.** When the number of users increases, computation time can increase considerably, especially in correlation-based algorithms where correlation has to be computed for every pair of users in the database.

It is clear that there is a need for an alternative system that can dynamically adapt to changes in user preferences and is at the same time scalable and fast to respond. By using adaptive algorithms (such as evolutionary algorithms and swarm intelligence), this thesis proposes that these problems can be overcome.
2.3. Adaptive Systems

John Holland, the founder of the field of genetic algorithms, has defined an adaptive system to be a system that has means of monitoring its own performance and can adjust efficiently to its environment (Goldberg 1989). In order to improve the capabilities of recommend systems, this thesis will focus on two types of adaptive algorithm: evolutionary algorithms and swarm intelligence.

2.3.1. Evolutionary Algorithms

Evolutionary algorithm is a term used to describe a population-based system that makes use of evolutionary processes such as natural selection or survival of the fittest to solve problems. More precisely, individual structures in a population evolve by means of random variation via mutation, recombination and other operators. Natural selection then takes place whereby the fittest tend to survive and reproduce, thus yielding potentially better offspring. It is, in fact, a method of searching among a large number of possibilities for solutions (Mitchell 1998). Evolutionary algorithms can be divided into four major categories:

- Genetic Algorithms (Holland 1975; Goldberg 1989)
- Genetic Programming (Koza 1992)
- Evolutionary Programming (Fogel et al. 1966)
- Evolution Strategies (Rechenberg 1973; Rechenberg 1994)

All are highly successful 'problem-solvers' used to evolve solutions for hundreds of different applications: everything from design optimization and robot control to music composition (Bentley 1999; Bentley and Corne 2001). Out of the four types, GAs are well documented and have been shown to work successfully well on a wide range of applications, especially optimisation problems (Bentley 1999; Goldberg 1989). Thus, only genetic algorithms will be employed in this thesis.
Genetic Algorithms (GAs)

Genetic algorithms were created by John Holland and his colleagues at the University of Michigan. In the mid-seventies, Holland published a book (Holland 1975) which started this field of study, describing theoretical foundations behind these algorithms.

Genetic algorithms are inspired by natural evolution. Populations of individuals or solutions are maintained, these are evaluated to find out how well each solution solves the problem, with better fitness scores given to better solutions. "Parents" are then selected from the population based on fitness – the fitter the solution, the more likely it is to become a parent. Child solutions are generated from the parents, employing crossover and mutation operators to ensure children resemble their parents with minor variations. The children replace the current population, the solutions are evaluated, parents are picked, and so on, for a number of generations. Evolution causes the solutions to improve until they satisfy the current problem (Bentley and Corne 2001).

GAs are today renowned for their ability to solve a huge variety of optimisation problems and for their consistent ability to provide excellent results (Holland 1975; Goldberg 1989; Davis 1991; Fogel 1994). A variety of GAs and variants exist; these include the canonical GA (Holland 1975; Goldberg 1989), steady state (Syswerda 1989), distributed (Whitley and Starkweather 1990), parallel (Adeli and Cheng 1994) and hybrid GAs with local search algorithms (Radcliffe and Surry 1994). Regardless of the type of GA employed, all GAs have a common algorithm. In GAs, there are two separate spaces: the search space and the solution space. The search space is a space of coded solutions to the problem, while the solution space is the space of actual solutions. In order to evaluate the quality or fitness of each solution, coded solutions or genotypes must be mapped onto actual solutions or phenotypes. In more detail, GAs comprise the following stages (figure 2.3):
Figure 2.3: The simple genetic algorithm

(1) Random Initialisation
GAs maintain a population of individuals; each individual consists of a genotype and its corresponding phenotype. Genotypes are usually in the form of a bit-string known as chromosome. A chromosome is normally divided into genes, where each gene can take different values known as alleles. Phenotypes, on the other hand, usually consist of collections of parameters which define the solution to a problem (for example, our DNA constitutes our genotype and our physical bodies constitute our phenotype). The simple GA starts by creating $n$ individuals where the genotype of each individual is initialised with random alleles.
(2) Evaluation

The process of evaluation involves assessing the quality of each individual or candidate solution in the population. The phenotype of every individual is evaluated and assigned a fitness value according to how well it satisfies the problem objective or fitness function.

(3) Reproduction or Selection

A selection process is required in order to pick relatively fit individuals from the population to act as parents (to be used in the reproduction of offspring for the next generation). This is considered a very important process as selection influences the performance of future generations. The strategy in which individuals are selected has to be carefully chosen as it can also introduce problems such as premature convergence. There are many different selection strategies; the most commonly used three are fitness-proportionate, ranking and tournament (Goldberg 1989).

- **Fitness proportionate**: this is the most popular selection scheme out of the three. It involves selecting individuals according to their fitness, relative to the average fitness of the population. An example of this method of selection is the roulette wheel. Each individual is allocated a slot on the wheel where its size is proportional to its fitness. The assumption is that those with higher fitness have a higher probability of being picked. However, there is a problem associated with using the fitness proportionate scheme; there is a chance that an individual with a high fitness value may get picked frequently which reduces diversity in the population and results in premature convergence upon a local maxima or minima (Goldberg 1989).

- **Ranking**: to solve the problem of premature convergence, Baker (1985) proposed ranking selection. The method involves sorting the population in order of raw fitness which does not introduce bias towards one or two superior individuals. Selection is then based on rank.

- **Tournament**: there are a number of variations of tournament selection (Goldberg and Deb 1991). The simplest one involves randomly selecting t individuals from the population and placing them in a competition pool. These selected individuals then compete and the winner (with highest fitness) is chosen as parent. The typical value of t is 2 (binary tournament).
Elitism is a strategy used in conjunction with one of the other selection schemes (de Jong 1975). It was proposed by de Jong that best individual(s) should be maintained in the next population. This is to ensure that the best fitness is maintained and does not get worse over time.

(4) Crossover and Mutation

Crossover and mutation are two very important GA operators. Crossover is a process of creating new offspring from a selected pair of parents. The most commonly used is the 1-point crossover where a position $k$ between 1 and the string length is randomly chosen. Two new offspring for the new generation are generated by swapping all bits after the crossover point. Figure 2.4 illustrates the process of performing a 1-point cross over at position 5. Other types of crossover include 2-point and uniform. Mutation is used to make small alterations to the offspring by randomly flipping bits in the string. It introduces variations into a population (Goldberg 1989). According to Goldberg, mutation rates should be small and hence, on the order of one mutation per 1000 bit transfers i.e. 0.001 per bit. However, de Jong found that mutation rate should be set inversely proportional to the population size (de Jong 1975), hence for a population size of 30, a mutation rate should be 0.033.

![Figure 2.4: 1-point crossover at position 5.](image)

2.3.2. Swarm Intelligence

By observing behaviour of social animals such as birds, ants and fish, a number of scientists have created algorithms that simulate this biological nature which can in turn be used to solve complex real world problems (Kennedy and Eberhart 1995). When these individuals form a group or a herd or a swarm or a flock or a school, remarkable collective intelligent behaviour occurs. Yet, there is no central force controlling the
interaction between them and each individual only exhibits limited local perception and intelligence. Consider the activities of a colony of ants building a nest or searching for food, a flock of birds flying in a well-choreographed way, or a school of fish swimming together avoiding predators (Bonabeau et al. 1999).

In *Swarm Intelligence: From Natural to Artificial Systems*, Bonabeau et al. (1999) define swarm intelligence as "the emergent collective intelligence of groups of simple agents". Kennedy et al. (2001) agree with the definition, but prefer not to tie swarm intelligence to the concept of "agents" as members of a swarm fall short of "autonomy" and "specialisation" characteristics of agents. According to their book, swarm intelligence is defined as "a population of interacting elements which is able to optimise some global objective through collaborative search of a space".

Millonas (1994) has articulated five basic principles of swarm intelligence. These are:

- The *proximity* principle: the population should be able to carry out simple space and time computations.
- The *quality* principle: the population should be able to respond to quality factors in the environment.
- The principle of *diverse response*: the population should not commit its activity along excessively narrow channels.
- The principle of *stability*: the population should not change its mode of behaviours every time the environment changes.
- The principle of *adaptability*: the population must be able to change behaviour mode when it is worth the computational price.

Two main uses of swarm intelligence are identified in (Kennedy et al. 2001) as

- Intelligence and aspects of culture – this is to aid the study of how our own minds work by looking at the behaviours of other social animals. It was proposed that human intelligence should be done by modelling humans in a social context i.e. interacting with each other.
- Optimisation – the way in which animals work together or interact can be applied to solve various real-world problems in a more efficient way. This is described later in the Particle Swarm Optimisation section.
Three main types of swarm intelligence techniques are discussed below. However, the rest of this thesis focuses purely on the particle swarm optimisation approach.

**Flocks**

Through computer simulations, scientists discovered that these phenomena can be explained using ideas of self-organisation and de-centralisation (Bonabeau et al. 1999). A realistic simulation of bird flocking has been implemented by Craig Reynolds, who has gone on to provide animation of herds and flocks for animated movies such as the Wildebeest stampede scene from Walt Disney's The Lion King (Reynolds 1987). In his program, the simulated flocking creatures are referred to as *boids*. He proposed that the flocking birds/boids are driven by these 3 local forces or rules:

- **Collision Avoidance or Separation** – the boids try not to get too close to nearby flockmates.
- **Velocity Matching or Alignment** – the boids try to move at the same speed and in the same direction (velocity) as nearby flockmates.
- **Flock Centering or Cohesion** – the boids try to be at the centre of the local flockmates.

In this model, the movement of each boid is influenced only by the local flockmates within a small neighbourhood or region around itself. The neighbourhood is defined by *distance*, measured from the centre of the boid, and *angle*, measured from the boid's direction of flight. Flockmates that are outside the neighbourhood are ignored. Figure 2.5 below shows a boid's neighbourhood adapted from Reynold's website[^1].

The flocking model is therefore an example of emergent behaviour, one of the main principles in the field of Artificial Life, where interaction between simple behaviours of individuals can give rise to more complex yet organised collective behaviour (Heppner 1997). Most applications based on this model are computer animation and behavioural simulation (Reynolds 1988).

**Particle Swarms**

In addition to Reynold's rules, Frank Heppner (Heppner and Grenander 1990; Heppner 1997), a zoologist from University of Rhode Island also came up with a further rule, which stated:

- Attraction to a "roost" or "target" – this is a target position to which the birds are attracted.

Unlike Reynolds's boids, Heppner's are attracted to a roost; the closer they get to the target, the stronger the attraction would become. In his system, Heppner's birds maintain a target velocity but occasionally, they would deviate from the intended course by the influence of a random factor such as a gust of wind. This results in a realistic decentralised simulation of bird flocking choreography (Kennedy et al. 2001).

This "roost" is a dynamic force that has been introduced into the system, enabling members of the flock to utilise one another's knowledge (Kennedy and Eberhart 1995). It is this "attractive force" that inspired Kennedy and Eberhart to invent the particle swarm algorithm (Eberhart and Kennedy 1995; Kennedy and Eberhart 1995). Based on a social-
psychological metaphor (Kennedy 1997), the flock is modelled as a particle swarm system where each bird is now a mass-less particle flying in a search space trying to get to the target (solution). The velocity matching rule is eliminated and each individual particle now has a velocity which is dynamically adjusted according to its own and its neighbours’ flying experiences (Shi and Eberhart 1999).

**The Particle Swarm Optimisation algorithm**

Similar to genetic algorithms, particle swarm optimisation (PSO) is a population-based evolutionary technique that exhibits self-organisation and searches the solution space for optima through iterative procedures. However, it differs in that each individual (commonly referred to as particle) contains a position and a velocity ($n$-dimensional vectors). The velocity is responsible for changing the position of the particle to explore the space of all possible solutions, instead of using existing solutions to reproduce (Blackwell and Bentley 2002a). 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. Because of the rules governing the swarming process, particles will eventually swarm around the area in the space where fittest solutions are. Unlike GAs, the selection task is not performed in PSO; all particles are kept in the population throughout the run (defined as the number of iterations prior to termination) (Eberhart and Shi 1998b; Angeline 1998b). Thus, the survival of the fittest theory does not apply to PSO (Shi and Eberhart 1999).

PSO is becoming popular and has been successfully applied to solve a wide range of optimisation problems in many areas such as Reactive Power and Voltage Control (Fukuyama and Yoshida 2001), Artificial Neural Network Training (Kennedy and Eberhart 1995; Eberhart and Shi 1998a; Engelbrecht and Ismail 1999) and Electromagnetics (Ciuprina et al. 2002). Perhaps more interestingly, it has recently been used to improvise music (Blackwell and Bentley 2002b). *SWARMUSIC*, featured at the festival 'Music and the Mind' 2003, 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 correspond to 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 (Blackwell 2001; Blackwell and Bentley 2002b; Blackwell 2003a).

A PSO algorithm begins by initialising a population or swarm of particles which can be represented by either real-valued vectors or binary strings. The binary particle swarm approach is conceptually simpler and works well in multivariate decision making (Kennedy and Eberhart 1997; Kennedy et al. 1998). This thesis, however, employs the more commonly used real-valued version, therefore only this method will be discussed here. The underlying socio-cognitive theory presented in (Kennedy 1997) suggests that particles are influenced by their own previous success (known as the cognitive part) and also by the success of any particle in their neighbourhood (the social part).

Individuals can be connected to one another in various neighbourhood schemes. According to Kennedy's book, *Swarm Intelligence* (Kennedy et al. 2001), the two most common ways that individual particles can be topologically connected are:

- The *lbest* (local best) neighbourhood
  Each individual's neighbourhood contains itself and its $k$ adjacent neighbours (Eberhart et al. 1996). For example, given $k$ is equal to 2, the ring topology of this neighbourhood is shown below in figure 2.6.

![Figure 2.6: The lbest topology where $k = 2$.](image-url)
• The \textit{gbest} (global best) neighbourhood

All individuals in the gbest neighbourhood are connected and each particle is influenced by the best performance of any individual in the entire population (Eberhart et al. 1996). Figure 2.7 below shows the star topology of this neighbourhood.

![The gbest topology.](image)

Thus, the velocity and in turn the position of the particles are updated at every iteration according to two best positions: \textit{pbest} (personal best) and \textit{gbest} (or \textit{lbest}, depending on the chosen neighbourhood topology). The \textit{pbest} represents each particle's own previous best position whose fitness score is also stored. The gbest (or \textit{lbest}) position is the best position attained by any member in the swarm (or those that are in the particle's \textit{lbest} neighbourhood). Figure 2.8 shows the PSO algorithm. The particle update algorithm used in this thesis to modify particle velocities and positions is governed by these three rules:

\[
\begin{align*}
V_i &= wV_i + C_1p_i(X_{pbest,i} - X_i) + C_2r(X_{gbest} - X_i) \\
\text{if } (|V_i| > v_{\text{max}}) &\quad V_i = (v_{\text{max}} / |V_i|)V_i \\
X_i &= X_i + V_i
\end{align*}
\]

\hspace{1cm}\text{Rule 1} \hspace{1cm} \text{Rule 2} \hspace{1cm} \text{Rule 3}

where:

- \(x_i\) is the current position of particle \(i\)
- \(x_{pbest,i}\) is the best position attained by particle \(i\)
- \(x_{gbest}\) is the swarm's global best position
- \(v_i\) is the velocity of particle \(i\)
- \(v_{\text{max}}\) is the maximum velocity
- \(w\) is a random inertia weight between 0.5 and 1 (Eberhart and Shi 2001b)
- \(c_1\) and \(c_2\) are spring constants whose values are set to 1.494 (Eberhart and Shi 2001b)
- \(r_1\) and \(r_2\) are random numbers between 0 and 1 (Blackwell and Bentley 2002a)
INITIALISE the swarm with random initial positions
SET each particle's $p_{best}$ to be the initial position
SET one $p_{best}$ that returns the best fitness score to be the swarm's $g_{best}$
LOOP
  UPDATE velocity of each particle according to Rule 1
  CHECK velocity limit according to Rule 2
  UPDATE position of each particle according to Rule 3
  If the fitness of the new position is better than the particle's $p_{best}$ fitness
    Then REPLACE $p_{best}$ position with the new position
  If the fitness of the particle's $p_{best}$ is better than the swarm's $g_{best}$ fitness
    Then REPLACE $g_{best}$ position with this particle's $p_{best}$ position
UNTIL either the fitness of $g_{best}$ is below a threshold value or the maximum number of iterations is reached

Figure 2.8: PSO algorithm

This algorithm extends the original that was proposed by Kennedy and Eberhart (1995) whereby the notions of inertia weight, $w$, and velocity clamping (Rule 2) are added.

The inertia weight, $w$, was introduced by Shi and Eberhart (1998a; 1998b) to keep the balance between local and global search. A large inertia weight aids a global search while a small weight facilitates a local search. In their paper (Eberhart and Shi 2001a) it was proposed that in a non-dynamic environment, $w$ should be initially set to 1 and dynamically reduced to near zero during the course of a run. This allows the search space to be fully explored, thus possible good solutions can be found. However, in a dynamic case, it is not possible to predict the level or exploitation and exploration required at any given time. For this reason, a random inertia weight between 0.5 and 1 was proposed by Eberhart and Shi (2001b) as this gives the mean of 0.75 which is close to Clerc's constriction factor (described below) of 0.729 (Clerc 1999). They also chose a value of 1.494 for the spring constants $c_1$ and $c_2$ which again agrees with Clerc's analysis on convergence (Clerc 1999) and thus, provides a balance between exploitation and exploration (Blackwell and Bentley 2002a).

Clerc (1999) introduced a constriction factor which defines the resolution of the search by constraining and controlling velocities. Eberhart and Shi found that by combining the constriction factor with constraints on the maximum velocity, $V_{max}$, the PSO performance
can be improved significantly (Eberhart and Shi 2000). In this work, rule 2 has been applied to restrict the velocity to the range $[-v_{\text{max}}, v_{\text{max}}]$. This particular rule was taken from (Blackwell and Bentley 2002a) which implements spherically symmetric velocity clamping, instead of clamping the velocity to a box.

Other parameters associated with the PSO algorithm can also affect the performance. Several pieces of work have been carried out to investigate the effect that these parameters have and determine parameter settings that are suitable for an ‘off-the-shelf’ PSO (Carlisle and Dozier 2001). These parameters are:

- **Population Size** – the number of particles in the swarm.
  Shi and Eberhart (1999) found that population size has very little effect on the performance. Work by Carlisle and Dozier (Carlisle and Dozier 2001) also confirmed this; setting too large population sizes would result in a greater computation cost for each iteration. They suggested that a population of 30 particles is a good choice for most systems. On the other hand, Van den Bergh et al. proposed a Cooperative Particle Swarm Optimiser (CPSO), a variation of the standard PSO, which employs multiple swarms where each swarm handles a part of the solution vector being optimised. They proposed that a population/swarm size of 10 particles is usually sufficient (van den Bergh and Engelbrecht 2001). However, for more difficult problems, it is obvious that the population size needs to be increased accordingly.

- **Neighbourhood Size** – the number of influencing particles to which each particle is topologically connected.
  Experiments suggested that gbest populations tend to converge more rapidly on optima than lbest populations when they converge, but are also more susceptible to convergence on local optima (Eberhart and Kennedy 1995; Peer et al. 2003). Work by Suganthan (1999) involved gradually increasing the neighbourhood size (moving from lbest to gbest). However, their results were encouraging but inconclusive. Carlisle and Dozier (Carlisle and Dozier 2001) also conducted experiments by varying the neighbourhood size from 2 to global. They concluded that the gbest neighbourhood appears to be generally a better choice.
More recently, Kennedy and Mendes (2002) presented other neighbourhood topologies including the Von Neumann topology where the particles are connected in a grid network structure and each particle has 4 neighbours (above, below, to the left and to the right). It was shown that different neighbourhood topologies can affect performance, especially for multi-modal problems (Kennedy et al. 1998).

- **Particle Update Methods – Asynchronous and Synchronous.**

In an asynchronous update, the particle velocities and positions are updated in turn. This then allows the recent discoveries or knowledge to be utilised immediately and thus, solutions are usually found faster (Carlisle and Dozier 2001). Conversely, a synchronous method requires the update for all particles to occur in parallel and the best position is chosen between iterations. This update is usually associated with the gbest neighbourhood.

There have been several pieces of work that attempted to increase the performance of the original PSO algorithm by adjusting the particle dynamics. Kennedy and Eberhart themselves made a few adjustments to their particle update equations to improve performance (Kennedy et al. 2001). Interestingly, cluster analysis was used by Kennedy (2000) to modify the update equation which forces the particles to move towards the centre of their clusters, rather than being influenced by the gbest or lbest positions.

**PSO in dynamic environments**

PSO has been shown to be very successful in global optima in static environments such as the optimisation of a few benchmark functions (Eberhart and Shi 2001a). However, real world problems are not always static and the algorithm has to be able to adapt to these changes in these dynamic environments in order to achieve the best possible results. The optimisation of dynamic optima can be difficult for evolutionary algorithms due to diversity loss (Blackwell 2003c). There have been several pieces of work which applied evolutionary techniques to the dynamic problem (Angeline 1997; Bäck 1998; Branke 2001). Tracking changes in dynamic environments is still a new area for PSO and there have only been a relatively small number of studies in this area. Earlier work by both Eberhart and Shi (2001c) and Carlisle and Dozier (2000) showed encouraging results if a change in global optimum is small i.e. low spatial severity (Angeline 1997). However,
there was still a problem of over-specialisation which is a common problem for both evolutionary algorithms and PSO. To overcome this, both need to be able to adapt further to automatically detect change and respond to it (Blackwell 2003c).

Hu and Eberhart introduced 2 methods, \textit{changed-gbest-value} (Hu and Eberhart 2001) and \textit{fixed-gbest-value} (Hu and Eberhart 2002), for automatically detecting changes in dynamic systems. In the \textit{changed-gbest-value} approach, the global optimum value (gbest) is re-evaluated at each iteration. A change in the fitness score means that the evaluation function has changed. This method is based on the assumption that if the optimum location of a dynamic system changes, the optimum fitness of the current location also changes. However, this is not the case for all systems. The \textit{fixed-gbest-value} approach was therefore introduced. Here, the gbest value is monitored and if it has not changed for a number of iterations then there is a possible optimum change. In this method, the fixed duration number has to be chosen carefully. By setting it too small, false alarms can be raised for changes that do not exist. If the number is too large, the system is delayed in responding to changes. To prevent false alarms, it was proposed that the second best gbest value should also be monitored.

Blackwell and Bentley have published a number of papers on PSO in dynamic environments (Blackwell and Bentley 2002a; Blackwell and Bentley 2002c; Blackwell and Bentley 2002d; Blackwell 2003b; Blackwell 2003c and Blackwell 2003d). For their previously mentioned swarm music improvisation system (Blackwell and Bentley 2002b), it is required that the particle dynamics have to be able to provide a close tracking of a moving target (external musical event). The concept of fitness is removed and solutions are obtained by translating the shape of the swarm into melody. For this reason, it is important that the particles do not swarm too closely in order to avoid repetitive melody but at the same time a definite shape must be maintained if any musical coherence is to emerge (Blackwell 2001). For this to happen, they introduced dynamics similar to Reynold's boids simulation which consists of two opposing forces: attraction to the centre of mass and inter-particle repulsion (avoidance). Pbest and gbest positions in the original update equations are replaced by \( x_{centre} \) and \( x_{target} \). These two terms are the centres of mass of the swarm where \( x_{centre} \) is the centre of all particles and \( x_{target} \) is the centre of the captured target swarm. The avoidance force is zero if the separations between the two terms are greater than a threshold value, a specified level of repulsion is otherwise applied (Blackwell and Bentley 2002c). The results showed the algorithm successfully produced
a stable and ordered swarm of particles moving in an extended region around the dynamic target. This suggests that this algorithm with collision-avoidance may have a useful application as a dynamic search tool that can track and respond to target changes automatically.

Severity is a term, introduced by Angeline (1997), to characterise problems where the optimum position changes by a fixed amount $s$ at a given number of iterations (Blackwell 2003b). As mentioned earlier, most dynamic PSO systems have been shown to work well for low severity problems where the optimum position changes by small increments (Eberhart and Shi 2001c; Hu and Eberhart 2002). In an attempt to tackle the dynamic search problem with high severity, Blackwell and Bentley (Blackwell and Bentley 2002a) introduced the notion of charged particles which is based on an analogy of electrostatic energy. They have identified three types of particle swarm: neutral, atomic and fully charged. The neutral swarm is identical to the conventional PSO algorithm where the particles move towards the best position, with each moving with diminishing velocity around the best position. This allows good exploitation of the search space but diversity is lost. In the fully charged swarm, all particles are ‘charged’ making use of the collision-avoiding rule described earlier. The attraction force allows the particles to maintain an extended shape around the optimum position but at the same time, diversity is preserved due to the repulsion force. Thus, this type of swarm responds quickly to changes in environment. In the atomic swarm, 50% of the particles are charged and the other 50% are neutral. Because this type of swarm makes use of both neutral and fully charged models, it provides a good balance between exploitation and exploration of search space. Experiments were conducted with various levels of spatial severity and found that charged swarms performed best in extreme cases with high severity, whereas neutral swarms are better optimizers in milder environment i.e. static or small changes (Blackwell 2003d).

**Comparison between PSO and GAs**

There have been a number of attempts to compare the performance of PSO and GAs (Angeline 1998a; Eberhart and Shi 1998b; Kennedy et al. 1998). These works unanimously found that PSO generally reaches an optimal solution faster than the GAs. Angeline (1998a), however, points out that although PSO works very well to discover good solutions in earlier iterations, it does not possess the ability to improve upon the
quality of these solutions. Both Angeline and Eberhart et al. (Eberhart and Shi 1998b) suggest that by combining the two algorithms, this problem can be overcome. Several hybrid models have been implemented and shown some encouraging results. These models include employing selection mechanism in PSO (Angeline 1998b), introducing the concept of reproducing new particles (Løvbjerg et al. 2002) and using the idea of mass extinction in PSO (Xie et al. 2002).

**Ant Colony**

Inspired by the behaviour of real ant colonies, ant algorithms were invented as a computational approach to problem solving (Dorigo and Di Caro 1999). Entomologists have found that cooperation at the colony level is largely self-organised and arises from interactions amongst individual ants. Simple actions such as following the trail left by other ants can lead to various difficult problems being solved. From their earlier experiments, Deneubourg and Goss (1989) discovered that individual ants lay a certain amount of chemical substance called *pheromone* when foraging. Additionally, each ant prefers to follow a direction rich in pheromone rather than a poor one. Thus, this simple behaviour results in the discovery of the shortest path between a food source and their nest (Bonabeau et al. 1999). Figure 2.9 illustrates an example of two ants laying pheromone trails presented by Bonabeau and Théraulaz (Bonabeau and Théraulaz 2000).

![Ant Colony Diagram](image)

**Figure 2.9:** The top diagram shows two ants leaving the nest at the same time. The bottom diagram shows that each ant has taken a different route (the pheromone trails are represented by the dotted lines). The ant that took the shorter path returns first and the amount of pheromones left on this path is now twice as much. Thus, other ants are more attracted to this shorter path than the longer one.
Dorigo and Di Caro (1999) introduced ant colony optimisation (ACO) algorithm which applies various behaviours of ant colonies to solve optimization problems such as rerouting of network traffic in busy telecommunication systems. The underlying idea for this algorithm is that good solutions (usually corresponding to paths between the food source and the nest) are reinforced by a virtual pheromone trail laid by individual artificial ants (de Castro 2002). Because pheromones evaporate over time, this allows the ants to explore new paths.

ACO has successfully been used to solve the classic travelling salesman problem where the shortest path is required to visit a given number of cities (Dorigo and Gambardella 1997a; Dorigo and Gambardella 1997b). One advantage of ACO is its adaptability. Because the artificial ants are continuously exploring different paths, backup solutions are provided by the pheromone trails. In the event that a link of the solution path becomes infeasible (due to unforeseen circumstances), a set of alternatives already exists.

The ACO algorithm using reinforcement and evaporation mechanism has also been employed to solve other problems include scheduling tasks in a factory (Lloyd 2001) and load balancing in telephone networks (Schoonderwoerd et al. 1997).

There have been various pieces of work based on other collective behaviours found in ant colonies (Bonabeau and Théraulaz 2000). The decentralised effort by ant workers to transport a large food item has led to more effective algorithms for robots (Kube and Zhang 1992; Kube and Bonabeau 2000). Division of labour can also be applied to distributed problem solving (Larsen 2001). More interestingly, in some ant species, ant workers sort their larvae or cluster their dead to clean the nest. These clustering behaviours have been used to implement sorting and data clustering algorithms for analysing financial data (Lumer and Faieta 1994; Bonabeau et al. 1999; Ramos and Abraham 2003). In particular, Lumer and Faieta (1994) devised a clustering method for exploring a large database of bank customers. Their approach is based on the way in which the leptothorax unifasciatus ant workers sort the colony’s brood where similar items such as eggs and microlarvae are placed together (Bonabeau and Théraulaz 2000). The problem was used to determine which of its customers is likely to repay a loan. Traditional classification methods could not be used here as most of the customers have not borrowed money from any financial institutions. In order to solve this problem,
Lumer and Faieta represented each customer as a point in a plane where each point is treated like a brood item. Artificial ants can then move the customers around, sorting them according to surrounding items. The distance between two customers indicates the similarity between them. All characteristics of the customers are considered by the software ants simultaneously in order to make their sorting decisions. As a result, the algorithm presents visualised clusters of customers with similar characteristics which are then used by loan officers to predict more accurately if a customer is likely to repay a loan.

2.3.3. Summary of Adaptive Systems

Evolutionary algorithms and swarm intelligence have been shown to be possible solutions to the problems identified in section 2.2. As both approaches are adaptive, they can be used to learn user preferences and continuously fine-tune themselves to any changes which may occur. From the field of evolutionary algorithms, this thesis focuses on the genetic algorithms. GAs are renowned for their ability to solve a myriad of large-scale and complex optimisation problems and for their consistent ability to provide excellent results quickly (Holland 1975; Goldberg 1989; Davis 1991; Fogel 1994). From the field of swarm intelligence, the thesis focuses on particle swarm optimisation. More recent research on PSO has shown that similar large-scale and complex problems can be addressed with good results and quicker in most cases.

2.4. Summary

Section 2.2 investigated all related work in the field of recommender systems. By reviewing different approaches taken by past and current systems (the majority of which employ a collaborative filtering technique), a number of shortcomings were identified. The most significant shortcomings are

- slow response time
- nonadaptive
- unscalable

These then highlighted the need for an alternative approach that can dynamically adapt to changes in the user preferences. At the same time it should be scalable to allow for an increasing number of users yet remain quick to respond to users' requests.
Section 2.3 presented two potential solutions, evolutionary algorithms and swarm intelligence, which can be employed to address these shortcomings. Having examined both areas in detail, it was confirmed that they have desirable properties, including the necessary adaptive capabilities, and have not been used for the purpose of providing recommendations.
CHAPTER 3

Evolving Good Recommendations with a Genetic Algorithm

The previous chapter highlighted various deficiencies in current recommender systems, and the need to make recommender systems more adaptive. This chapter describes a new recommender system, which employs a genetic algorithm to learn personal preferences of users and provide tailored suggestions. It focuses on the use of evolutionary search 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. The chapter is organised as follows: section 3.1 outlines the MovieLens dataset used for experiments, and section 3.2 describes the recommender system and genetic algorithm. Section 3.3 provides experimental results and analysis. Finally section 3.4 concludes.

3.1. MovieLens Dataset

The dataset collected through the MovieLens website (http://www.movielenls.umn.edu) has been made available for research purposes and is used in this research. The database 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 rating of a movie 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. The evolutionary recommender system uses 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 and western. Figure 3.1 shows the relationships and dependencies between the various elements in the database and table 3.1 presents a description of each table in the database.
Figure 3.1: Relationships and dependencies between the various elements in the database
3.1.1. Composition of Data

Analysis of the dataset reveals the following:

The pie chart in figure 3.2 shows the gender split of MovieLens users where 71% are male and 29% are female.
The pie chart in figure 3.3 shows the age groups of MovieLens users where the majority of users (35%) fall within the 21-30 age group and only 2% are over 60.

Figure 3.4: Genres
The pie chart in figure 3.4 shows the breakdown of movie items by genre. The first genre classified as 'unknown' represents 0.18% of the total as only 3 out of 1682 items fall under this category. A quarter of the movies featured belong to the drama genre.

![Pie chart showing genre breakdown](image)

Figure 3.5: Occupations

The pie chart in figure 3.5 shows the percentage of users belonging to the various occupations. The majority of users are students.

Note that correlation of data is not addressed in this thesis but may be important for large datasets.

3.1.2. Preparation of Training and Test Data

In order to evaluate the performance of each system, for all experiments in this thesis, the MovieLens data was divided into 2 datasets: a training set and a test set. For each user, 30% of all movie items that the user had rated were randomly selected to be in the training set, with the remaining 70% forming the test set. It is vital that the system has to be able to predict with a reasonable level of accuracy even for those users who have rated a small number of movies. The training/test ratio of 30:70 was thought to be suitable as 30% of the minimum number of movies required (20 movies) for each active user is 6,
this will therefore allow the performance of the system to be observed for users with the number of ratings of 6 or more. Note that the experiments in section 6.8 (see appendix D) will instead use data gathered from the pilot study.

The training set is used by each system to learn each user's preferences and in turn determine a group of users similar to that user. These similar users are then used to predict a rating that the user would give for each item in the test set. The performance of the system is a measure of how closely the predicted ratings are to the actual ratings for all test items.

Figure 3.6 shows the number of training and test items for the first 50 users which are employed as “active users” (those users for which the system is providing recommendations) in the experiments. The number of items rated by each user varies considerably. In particular, user 13 had rated a very large number of 636 items (of which 190 is for training) compared to user 19 who had rated only 20 (6 for training). However, the difference in the number of items rated has little effect on performance as it is possible that the items rated by user 13 may not have been rated by anyone else. The number of items would only play a big part in the performance if these items had also been rated by other users.

3.2. GA Recommender System

The system described in this chapter is based around a collaborative filtering approach, building up profiles of each user and then using an algorithm to find profiles similar to the current user. The current user is referred to 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 (see ‘Shortcomings of
Collaborative Filtering Systems' in chapter 2), the task of finding appropriate similarities is often difficult. To overcome these problems, existing 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 an evolutionary algorithm to the problem of profile matching.

The full algorithm is shown below in figure 3.7.

```
LOOP 1 for each active user, A
    Create profile(A)
LOOP 2
    LOOP 3 for each individual in population
       Create an empty neighbourhood for A
       Map genotype to phenotype (a set of feature weights, w)
       LOOP 4 for each user j where j ≠ A and j is selected from current Profile Selection set
          Compute similarity value between A and j, euclidean(A,j) with w
          if similarity value is less than similarity threshold
          Add j to A's neighbourhood
       END LOOP 4
    LOOP 5 for each item i in training set for A
       Compute the predicted rating for i, predict_rating(A,i), with A's neighbourhood
    END LOOP 5
    Set fitness of individual to be the mean difference between predicted and actual ratings
    END LOOP 3
    Rank individuals in population according to fitness score
    if fitness score of best individual is less than solution threshold
       solution is found, go to line 39
    else
       Create a new (empty) population
       LOOP 6 for each individual, starting from the best
          Make a copy of individual and add it to new population
          if individual's lifespan is less than maximum lifespan threshold
             Increment lifespan of individual by 1
          else
             Set a flag to remove individual in next generation
       END LOOP 6
       UNTIL a quarter of new population is filled (END LOOP 7)
    Replace current population with new population
    Increment the number of generations by 1
END LOOP 2
UNTIL solution found or maximum number of generations is reached (END LOOP 2)
if solution found
   Set A's final feature weights to be the phenotype of the solution
else
   Set A's final feature weights to be the phenotype of the best individual
   LOOP 8 for each user j where j ≠ A and j is selected from current Profile Selection set
      Compute similarity value between A and j using A's final feature weights
      if similarity value is less than similarity threshold
      Add j to A's final neighbourhood
   END LOOP 8
   LOOP 9 for each item i in test set for A
      Compute the predicted rating for i using A's final neighbourhood
      if the difference between predicted and actual ratings for i is 0
         Increment the number of correct predictions (zero tolerance)
      if the difference between predicted and actual ratings for i is either 1 or 0
         Increment the number of correct predictions (at-most-one tolerance)
   END LOOP 9
```

Figure 3.7: The algorithm describing the GA Recommender System

From figure 3.7 above, the algorithm starts by creating a profile of an active user, A. This task in line 2 is referred to as the profile generator, see section 3.2.1. Lines 3 to 42 represent the learning process where the genetic algorithm (described later in 3.2.4) is
responsible for evolving a set of feature weights that reflects A's recommendation preferences. Users that participate in each run are selected using a method of profile selection, see Neighbourhood Selection section below, to form the current profile selection set (lines 8 and 44). The neighbourhood selection process, described later in 3.2.2, is employed to select the neighbourhood of users similar to A (loop 4 from line 8 to line 12 and loop 8 from line 44 to line 48). Once the final neighbourhood for A is found, it is used to predict ratings for the movie items in the test set for A, refer to loop 9 from lines 49 to 55. The accuracy of the predictions is then recorded to evaluate the performance of the system, see Experiments section. Note that movie items which receive a high predicted rating (either 4 or 5) can then be recommended to the user, see section 3.2.3. Each of these sections is described in more detail below.

3.2.1. 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 figure 3.8.

<table>
<thead>
<tr>
<th>Rating</th>
<th>Age</th>
<th>Gender</th>
<th>Occupation</th>
<th>18 Genre frequencies</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>23</td>
<td>0</td>
<td>45</td>
<td>000000100010000000</td>
</tr>
</tbody>
</table>

Figure 3.8: 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.
3.2.2. Neighbourhood Selection

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

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. These values of $n$ were chosen in order to allow the performance of the system to be observed when the number of users is low and ratings are perhaps sparse, as this is typical in the initial phase of deployment. As the number of users increases, there is a greater chance of finding more similar users and hence, the performance of the system usually increases.

Profile Matching

After profile selection, the profile matching process then computes the distances or similarities 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 evolutionary algorithm is used to fine-tune profile matching for each active user.
From the analysis of Breese et al. (1998), it seems that most existing recommender systems use standard algorithms that consider only "rating 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 similar 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 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 evolved by using a genetic algorithm.

A potential solution to the problem of evolving feature weights, \( w(A) \), for the active user, \( A \) is represented as a set of weights as shown below in Figure 3.9 where \( w_f \) is the weight associated with feature \( f \) whose genotype is a string of binary values.

<table>
<thead>
<tr>
<th>( w_1 )</th>
<th>( w_2 )</th>
<th>( w_3 )</th>
<th>...</th>
<th>( w_{22} )</th>
</tr>
</thead>
</table>

Figure 3.9: Phenotype of an individual in the population.

Each individual contains 22 genes, which are evolved by an elitist genetic algorithm (described later in section 3.2.4). The comparison between two profiles can now be conducted using a modified Euclidean distance function, which takes into account multiple features, see figure 3.10. \( \text{euclidean}(A, j) \), the similarity between active user \( A \) and user \( j \), is defined as:

\[
\text{euclidean}(A, j) = \frac{1}{z} \sum_{i=1}^{2z} \sqrt{\sum_{j=1}^{22} w_f \cdot \text{diff}_i(f(A, j))^2}
\]

where:
- \( A \) is the active user
- \( j \) is a user provided by the profile selection process, where \( j \neq A \)
- The common items that users \( A \) and \( j \) have rated are defined as the set \( \lambda_1 \ldots \lambda_z \).
- \( z \) is the number of common movies.
$w_f$ is the active user's weight for feature $f$.

$i$ is a common movie item, where $\text{profile}(A,i)$ and $\text{profile}(j,i)$ exists.

$\text{diff}(A,j)$ 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.

\[ \text{euclidean}(A,j) = \text{similarity}(A,j) \]

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{diagram}
\caption{Calculating the similarity between $A$ and $j$}
\end{figure}

**Best Profile Collection**

Once the Euclidean distances, $\text{euclidean}(A,j)$, have been found between $\text{profile}(A)$ and $\text{profile}(j)$ for all values of $j$ picked by the profile selection process, the best profile collection algorithm is called. This ranks every $\text{profile}(j)$ according to its similarity to $\text{profile}(A)$. 

85
profile(A). The value returned by the Euclidean function lies in the range of 0 to 1, with 0 representing a perfect match to the active user and 1 being the least similar. However, it makes sense that users who are more similar to the active user should provide a greater contribution than less similar ones in the predicted rating equation (described later in section 3.2.4). Therefore, the calculated Euclidean distances have to be reversed i.e. 0 maps to 1 and 1 maps to 0. The function used to do this is simply, \( f(x) = 1-x \). The system then 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.

3.2.3. Making a 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 evolved weighting values), movies that these users like have a reasonable probability of being liked by \( A \).

3.2.4. Genetic Algorithm

As described in section 3.2.2, a genetic algorithm has been used to evolve feature weights for the active user, and hence help tailor the matching function to the user's specific personality and tastes.

An elitist genetic algorithm was chosen for this task, where a quarter of the best individuals in the population are kept for the next generation. When creating a new generation, individuals are selected randomly out of the top 40% of the whole population to be parents. Two offspring are produced from every pair of parents, using single-point crossover with probability 1.0. Mutation is applied to each locus in genotype with probability 0.01. A simple unsigned binary genetic encoding is used in the implementation, using 8 bits for each of the 22 genes. The GA begins with random genotypes.
A genotype is mapped to a phenotype (a set of feature weights) by converting the alleles of the binary genes to decimal. The feature weights can then be calculated from these real 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 to be 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 phenotype is then calculated by summing the real values for all 22 features. 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 1.

Fitness function
Calculating the fitness for this application is not trivial. Every set of weights in the GA population 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 set of weights, in order to calculate its fitness.

But running a recommender system only produces recommendations (or predictions), not fitnesses. A poor set of weights might result in a poor neighbourhood set of profiles for the active user, and hence poor recommendations. A good set of weights should result in a good neighbourhood set, and good recommendations. So a method of calculating the quality of the recommendations is required, in order that a fitness score can be assigned to the corresponding weights.

One solution would be to employ the active user as a fitness function. This would involve obtaining feedback from the user by asking him to judge the quality of recommendations (Bentley and Corne 2001). His input could be used to help derive fitness scores for the current set of feature weights. This fitness score would give a highly accurate view of the user’s preferences. However, it is unlikely that every user will be willing to participate in every recommendation – this would require far too much time for most users.

Instead, 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, if those users thought the movie was "average", then it is likely that the active user may 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 view it).

The predicted rating computation used in this paper has been taken from (Breese et al. 1998) and modified such that the Euclidean distance function (section 3.2.2) now replaces the weight in the original equation. The predicted rating, \( \text{predicted\_rating}(A, i) \), for \( A \) on item \( i \), can be defined as:

\[
\text{predicted\_rating}(A, i) = \text{mean}_A + k \sum_{j=1}^{n} \text{euclidean}(A, j) (\text{rating}(j, i) - \text{mean}_j) \quad [1, \text{p.33}]
\]

where:

- \( \text{mean}_j \) is the mean rating for user \( j \)
- \( k \) is a normalising factor such that the sum of the Euclidean distances is equal to 1 and thus,
  \[
  k = \frac{1}{\sum \text{euclidean}(A, j)}
  \]
- \( \text{rating}(j, i) \) is the actual rating that user \( j \) has given on item \( i \), if \( \text{rating}(j, i) \) exists.
- \( n \) is the size of the neighbourhood.

To calculate a fitness measure for an evolved set of weights, the recommender system finds a set of neighbourhood profiles for the active user, as described in section 3.2.2. 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 ratings of all items in the training set are used as a fitness score to guide future generations of weight evolution, see figure 3.11. Note that in the case where no other user is similar to the active user (the neighbourhood set is empty) or none of the users in the neighbourhood set has rated item \( i \), the predicted rating for \( i \) is simply the active user's mean rating.
3.3. Experiments

Four sets of experiments were designed to observe the difference in performance between the GA recommender system and a standard, non-adaptive recommender system based on the Pearson algorithm (Breese et al. 1998). In each set of experiments, the predicted ratings of all movie items in the test set (the items that the active user has rated but were not used in weights evolution) were computed using the final feature weights for that run. These ratings were then compared against those produced from the simple Pearson algorithm.

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 in section 3.2.2; all other details remain
the same.

\[
correlation(A, j) = \frac{\sum_{i=1}^{n} (\text{rating}(A, i) - \text{mean}_A)(\text{rating}(j, i) - \text{mean}_j)}{\sqrt{\sum_{i=1}^{n} (\text{rating}(A, i) - \text{mean}_A)^2 (\text{rating}(j, i) - \text{mean}_j)^2}}
\]

[2, p.33]

The four experiments also 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.

Table 3.2 below shows the experimental parameters. Note that preliminary experiments
were performed and the following set of parameter values was found to be most suitable.
These values were kept the same in all four experiments:

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>population size (The number of individuals in the population at each generation)</td>
<td>75</td>
</tr>
<tr>
<td>termination threshold (When the fitness score of the best individual (set of feature weights) is below the threshold, a good solution is found and this set of weights is used as the final result for the current run)</td>
<td>0.06</td>
</tr>
<tr>
<td>maximum number of generations for each run (If the number of generations reaches this value and a solution has not been found, the best individual for that generation is used as the final result)</td>
<td>300</td>
</tr>
<tr>
<td>weight reduction size (The scaling factor for the 18 genre frequencies)</td>
<td>4</td>
</tr>
<tr>
<td>number of runs (The number of times the system was run for each active user)</td>
<td>30</td>
</tr>
</tbody>
</table>

Table 3.2: Experimental Parameters

Note that in this thesis, the value of \( k \) (users in the Pearson algorithm) is set to be a half of
the number of users in each experiment (for example, in experiment 1, \( k = 5 \)). This was
found in preliminary experiments to yield the greatest level of accuracy.
3.3.1. Experimental Setup

The four sets of experiments are as follows:

**Experiment 1:** Each of the first 10 users (out of 944) was chosen as the active user, $A$, in turn (loop1 starting at line 1 and ending at line 56 of figure 3.7). The first 10 users formed the Profile Selection set and were used to provide recommendations (loop 4 from line 8 to line 12 and loop 8 from line 44 to line 48 of figure 3.7).

**Experiment 2:** Each of the first 50 users was chosen as the active user, $A$, in turn. The first 50 users formed the Profile Selection set and were used to provide recommendations. Note that the 10 users in experiment 1 were a subset of the 50 users here.

**Experiment 3:** Each of the first 10 users was chosen as the active user, $A$, in turn. For each run, 9 different users were selected randomly (out of 944) to form the Profile Selection set and were used to provide recommendations.

**Experiment 4:** Each of the first 50 users was chosen as the active user, $A$, in turn. For each run, 49 different users were selected randomly (out of 944) to form the Profile Selection set and were used to provide recommendations. Note that the 9 users in experiment 3 were a subset of the 49 users here.

The results of these experiments were captured in loop 9 from lines 49 to 55 in figure 3.7.

This thesis uses 2 different metrics that evaluate the prediction accuracy of the system:

- **Zero tolerance** – the accuracy of the system is computed 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.

- **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 – the rating given to 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. Additionally, this metric represents a more realistic situation as the reliability
of the ratings given by user is not 100% accurate. This was illustrated by an experiment carried out by Hill et al. (1995) where users were asked to re-rate the same items after 6 weeks. Their results showed that the Pearson correlation between first-time and second-time ratings was 0.83.

This chapter presents only the results computed using zero tolerance. The results for at-most-one tolerance are presented and used to compare against the PSO system in chapter 4.

3.3.2. Results

![Graph](image-url)

Figure 3.12: Comparison between PA and GA - best results for experiment 1

![Graph](image-url)

Figure 3.13: Comparison between PA and GA – best results for experiment 2
Figures 3.12 to 3.15 show the results for experiments 1 to 4, respectively. Each graph shows 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. Whilst the predictions computed with the Pearson algorithm always remained the same given the same parameter values, those obtained from the GA varied 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 correct predictions) was chosen and plotted against the result from the Pearson algorithm. The best rather than average was plotted since this is closest to the real world scenario where this system could be run offline and the current best set of feature weights would be set as the initial preference of the active user. Following this, the evolved weights could be stored on the user’s local machine. A local copy of the system would then be responsible for fine-tuning the weights to suit that user’s preferences further. This way the processing load on the server would be reduced and parallelism could be achieved.
Figure 3.12 shows that in the first experiment, the GA recommender performed equally well (or better) compared to the Pearson algorithm on 8 active users out of 10.

Figure 3.13 shows that in the second experiment, out of the 50 users the accuracy for the GA recommender fell below that of the Pearson algorithm for 14 active users. In all other cases, the accuracy for the GA recommender was found to be better – in some cases the improvement was as great as 31%.

The random sampling for experiment 3 showed great improvement on the prediction accuracy for the GA recommender, see figure 3.14. All 10 active users performed better than the Pearson algorithm.

The results for the last experiment show that the accuracy for the GA recommender was significantly better for all but 4 active users, see figure 3.15. This suggests that random sampling can work very well and could be used as a good solution whenever the database is too large for all users to be considered.

For 10 active users, a typical run by the PA system took approximately 3 seconds which was significantly faster than that by the GA system of approximately 350 seconds. Similarly, for 50 active users, a typical run by the PA and GA systems took approximately 20 and 8310 seconds respectively. This shows that although the GA system has shown to be able to obtain a greater level of prediction accuracy than the PA system for most active users, computational speed of the GA needs to be improved dramatically in order for it to be used in an online system. Note that this problem will be addressed in following chapters.

3.3.3. Analysis of Results

Figure 3.12 indicates that the prediction accuracy for the active user 3 and 8 on the GA recommender was worse than that obtained from using the Pearson algorithm. But when the number of users was increased to 50 in experiment 2, the accuracy for the two mentioned active users rose and outperformed the other algorithm. This was expected – as the number of users goes up, the probability of finding a better matched profile increases and hence, the accuracy of the predictions should increase as well.
The accuracy patterns in both experiments 3 and 4 for the active users 1 to 10 look very similar. Both show an improved accuracy compared to the Pearson algorithm but in experiment 4 there seems to be a greater improvement. Again, this is likely to be because of the increase in the number of users. The results suggest that random sampling is a good choice for the profile selection task of retrieving profiles from the database. Random sampling was expected to be better than fixing which users to select because it allowed the search to consider a greater variety of profiles (potentially 10*30 runs = 300 users in experiment 3 and 50 * 30 = 1500 users in experiment 4) and hence, find a better set of well matched profiles.

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 preferences.

Looking at experiment 1, when more than one run for an active user achieved the same best performance (highest number of ratings being predicted correctly), results indicated that the same set of users had been selected for the neighbourhood to give recommendations. Moreover, for other runs that did not perform as well as the best run(s), users different to the ones that resulted in the best performance had been selected. For example, in all best runs for active user 2 in experiment 1, user 4 was always found to be present in the neighbourhood. Data gathered during experiment 2 corroborates this view. In addition, as the number of users was increased, the users that were originally selected for the neighbourhood in experiment 1 were still being chosen in experiment 2 as a subset of a larger neighbourhood. For example, as mentioned above, in experiment 1 user 2 picked user 4 to be in the neighbourhood, in experiment 2 this user picked users 4,13,18,22,42,43,49. This, however, only applies to the active users that performed better than the Pearson algorithm in experiment 1. The accuracy for active user 8 was worse in experiment 1, in which users 4, 5, 7 and 10 were selected. In experiment 2, the accuracy improved tremendously, as seen in figure 3.13, when users 4 and 10 were not included in the neighbourhood.

The trend described could not be observed when random sampling was used in experiments 3 and 4, as it was more difficult for the system to select the same users to examine at each run.
For each of the four experiments, figures 3.16 to 3.19 below show the average prediction accuracy for 30 runs. As mentioned earlier, the predictions computed with the Pearson algorithm always remain the same given the same parameter values and those obtained from the GA differ stochastically according to the feature weights of that run. By observing the average predictions, the overall system performance can be further analysed.

Figure 3.16: Comparison between PA and GA – average results for experiment 1

Figure 3.17: Comparison between PA and GA – average results for experiment 2
Figure 3.18 shows that in the first experiment, the averages of the predictions produced by the GA recommender were mostly the same as the best results shown earlier in figure 3.10. As there was little difference between the best and average results, this demonstrates that the GA system managed to find the best possible results in most cases.

In experiment 2, the average prediction by the GA recommender still outperformed that obtained by the Pearson algorithm on 22 active users, see figure 3.17. Those that fell below the predictions of the Pearson algorithm only did so by a small margin.

The results by the GA recommender for experiments 3 and 4 (see figures 3.18 and 3.19 respectively) display slightly worse average predictions than the ones found by the best runs (figures 3.14 and 3.15). This suggests that although random sampling works well in finding the best results, the results obtained from all runs can vary due to the randomness.
of the users chosen for that run. Again, the difference between the average predictions by
the GA and the best results by the Pearson algorithm was still small in most cases.

As mentioned earlier, the best runs would always be selected to give recommendations.
However, as shown above, the weights obtained in most runs by the GA algorithm would
still provide recommendations of acceptable accuracy.

For completeness, the worst results obtained by the four experiments are shown below in
figures 3.20 to 3.23.

![Figure 3.20: Comparison between PA and GA - worst results for experiment 1](image1)

![Figure 3.21: Comparison between PA and GA - worst results for experiment 2](image2)
It was found that the worst results obtained by the GA recommender for experiments 1 and 2 (figures 3.20 and 3.21) were still reasonable; the difference between them and those produced by the Pearson algorithm was small. However, this difference grew significantly when random sampling was used in experiments 3 and 4 (figures 3.22 and 3.23). This finding is important as it demonstrates that the weights play a vital role in the accuracy of the system. Wrong choice of weights can result in poor recommendations.

Looking at the final feature weights obtained for each active user, many interesting observations have been found. Here we focus on the first 2 experiments as they have 10 common users. Firstly, in experiment 2 when more than 1 run came up with the best performance, the feature weights seem to show very similar trends. For example, figure 3.24 below shows the weight emphasis on the first 2 features: rating and age. It is also clear that this user does not show any interest in the 3rd feature which is gender. So as
long as the people that are providing him recommendations have similar opinions and are in the same age group as him, he does not care whether they are male or female.

Figure 3.24: Feature weights from best runs for active user 2 (weights 5 to 22 are lower because of the scaling factor)

The feature weights obtained for active user 8 were also interesting, see figure 3.25. They show that for this user, age and gender (features 2 and 3) are more significant. By looking further at the movie genres (features 5-22), we found that people who have similar opinions as this user on action (feature 5), adventure (feature 6), horror (feature 15), romance (feature 18) and war (feature 21) movies are likely to be picked for the neighbourhood set. As these genres are stereotypically related to gender and age, for example, men prefer action movies and war movies, the weights showed consistent description of the user’s preference.

Figure 3.25: Feature weights from the best run for active user 8

Another example is active user 7 whose weights show strong feelings towards documentary, mystery, sci-fi and thriller genres and emphasis on age, see figure 3.26 below. This user is a 57 year old male which may explain reduced significance of children and romance genres. Also, we discovered that children and animation genre features usually have similar weights – this could be because these two genres are usually related i.e. most animation films are children’s films like Disney cartoons.
Figure 3.26: Feature weights for active user 7

From the observations above, we can see that age is often equally as important as rating, and in some cases more so. This shows that the theory behind the original collaborative filtering that only the rating information is required does not always hold. This is hardly surprising as everyday experience suggests that most people listen to the recommendations made by their friends who are most likely to be in the same age group as them.

In experiment 1, all active users seem to have similar feature weights as the ones obtained in experiment 2 apart from users 3, 8, 9 and 10. We divide our analysis of this into 2 parts. Firstly, users 3 and 8 performed worse than the Pearson algorithm in experiment 1. This was because the weights obtained did not describe the users realistically. As the number of users was increased in experiment 2, better weights could be captured and hence produced improved performance. Secondly, the weights for active users 9 and 10 did not display any useful information in experiment 2. This resulted in reduced performance for them compared to the original algorithm. But in experiment 1, the weights for these 2 users show a consistent trend resulting in increased accuracy compared to the Pearson algorithm in this experiment.

This approach has been shown to work well, but there are problems. As fitness scores are computed from the differences between the actual and predicted ratings, this is only achievable if the user has already actively rated movies, otherwise the intersection between recommended items and those already rated by the active user would return very few titles or even none. In this case, this approach will fail, as a fitness score cannot be determined.
In an earlier experiment with only 4 features: rating, age, gender and occupation, it was noticed that many solutions were found for items which are sometimes associated with gender (inferred by gender). For example, when the active user’s feature weights showed that the user preferred to be recommended by people of the same gender (3rd feature), solutions were often found for items that belonged to stereotyped genres. Figures 3.27 displays the feature weights obtained for a male user and table 3.3 lists recommended items for this user. 10 out of the 16 items (with a predicted rating of 4 or above) that were being recommended using this set of weights were action movies. This suggests that perhaps recommendations could be improved further by the use of association rules.

![Figure 3.27: Feature weights for active user 1 in 4-feature experiment](image)

<table>
<thead>
<tr>
<th>Film</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Braveheart</td>
<td>Yes</td>
</tr>
<tr>
<td>Apollo 13</td>
<td>Yes</td>
</tr>
<tr>
<td>Blade Runner</td>
<td>No</td>
</tr>
<tr>
<td>Aladdin</td>
<td>No</td>
</tr>
<tr>
<td>Independence Day (ID4)</td>
<td>Yes</td>
</tr>
<tr>
<td>Die Hard</td>
<td>Yes</td>
</tr>
<tr>
<td>Top Gun</td>
<td>Yes</td>
</tr>
<tr>
<td>Empire Strikes Back, The</td>
<td>Yes</td>
</tr>
<tr>
<td>Return of the Jedi</td>
<td>Yes</td>
</tr>
<tr>
<td>GoodFellas</td>
<td>No</td>
</tr>
<tr>
<td>Blues Brothers, The</td>
<td>Yes</td>
</tr>
<tr>
<td>Sting, The</td>
<td>No</td>
</tr>
<tr>
<td>Dead Poets Society</td>
<td>No</td>
</tr>
<tr>
<td>Star Trek: First Contact</td>
<td>Yes</td>
</tr>
<tr>
<td>Raising Arizona</td>
<td>No</td>
</tr>
<tr>
<td>Men in Black</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 3.3: recommended movies for active user 1
3.4. Summary

This work has shown how an evolutionary search can be employed to fine-tune a profile-matching algorithm within a recommender system, tailoring it to the preferences of individual users. This was achieved by reformulating the problem of making recommendations into a supervised learning task, enabling fitness scores to be computed by comparing predicted ratings with actual ratings. Experiments demonstrated that, compared to a non-adaptive approach, the evolutionary recommender system was able to successfully fine-tune the profile matching algorithm. This enabled the recommender system to make more accurate predictions, and hence better recommendations to users.

Additionally, these significant points were found:

- by increasing the number of users, the system performance can be significantly increased.
- random sampling is a preferred choice for the profile selection task of retrieving profiles whenever the database is too large for all users to be examined.
- the feature weights play a vital role in the accuracy of the system. Wrong choice of weights can result in poor neighbourhood sets, and hence poor recommendations.
- the feature weights obtained for each active user can be used to explain how recommendations are derived, helping to gain the users' confidence in the system.
- the results obtained from the experiments revealed interesting relationships amongst features, such as gender and genres where most male prefer action films.
- it was found that age is often equally as important as rating, and in some cases more so. This therefore shows that the theory behind the traditional collaborative filtering does not always hold.

Although the GA recommender system has shown to perform better than a non-adaptive approach in terms of prediction accuracy, the amount of time required to evolve an initial set of feature weights can be considerable if the number of users is large. Thus, there is still a need for a system that is fast and adaptive.
CHAPTER 4

A Recommender System using Particle Swarm Optimisation

This chapter investigates the use of particle swarm optimisation (PSO) in a recommender system. The chapter is organised as follows: section 4.1 compares a new PSO recommender system against the genetic algorithm and Pearson algorithm systems that were presented in chapter 3. Section 4.2 presents variations of the PSO recommender, and section 4.3 provides extensive analysis of the algorithm. Finally, section 4.4 concludes.

4.1. Comparison between PSO, GA and PA Recommender Systems

This section introduces a new recommender system, which employs a particle swarm optimisation algorithm to learn personal preferences of users and provide tailored suggestions. Experiments to assess performance of the system were performed and results obtained were compared against those of the GA and PA recommender systems.

4.1.1. Flying Geese PSO (FG_PSO) Recommender

To enable comparisons to be made, the PSO recommender has the same structure as the GA system described in chapter 3. The main difference between the two systems lies within the Profile Matching task, which is responsible for computing the distance or similarity between the selected profiles and the active user's profile. In this system, the task of capturing and fine-tuning the active user's feature weights, which was previously performed by a genetic algorithm, is now handled by a PSO technique. These weights are used to find a neighbourhood set of profiles similar to the active user, A. Movie items rated highly by the users in the neighbourhood set that the active user has not seen are then presented to him through a user interface. Because the neighbourhood set contains those users who are most similar to A (using the specific preferences of A through attained weighting values), movies that these users like have a reasonable probability of being liked by A. Figure 4.1 below shows the role of PSO within the Profile Matching
process where $A$ is the active user, $j$ is a user in the system and $i$ is a common item between $A$ and $j$.

$\text{euclidean}(A,j) = \similarity(A,j)$

Figure 4.1: Calculating the similarity between the active user $A$ and a user $j$.

**Particles**

The PSO recommender has a 22-dimensional swarming space where 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 GA recommender system). Similar to a population of individuals, the PSO employs a swarm of particles. Each particle $i$ has:

- a position, $x_i$. This is the current position of the particle within the swarming space and is represented by a vector with 22 components. It is randomly initialised at the start of each run.
The fitness of this position is computed using the same fitness function described in chapter 3. Each position is converted into a set of feature weights using the following formula.

\[ w(x_{i,f}) = \frac{x_{i,f}}{\sum_{f=0}^{22} x_{i,f}} \]  

where:

- \( x_{i,f} \) is the value of the \( f \)th component of the current position \( x_i \).
- \( w(x_{i,f}) \) is the corresponding feature weight of \( x_{i,f} \).

These weights are used to find a neighbourhood of profiles similar to the active user (using a Euclidean distance function previously described), which in turn are used to compute the predicted rating for all training movie items. These ratings are then compared against the actual ratings given by the active user. Finally, the average difference between the predicted and actual ratings is assigned to be the fitness score of the given position.

- a velocity, \( v_i \). This is the direction in which particle \( i \) moves and is also represented by a 22-dimensional vector.

- A personal best (pbest) position. This stores the position of the particle which yielded the best fitness value.

The particle with the highest fitness score out of all particles in each iteration is set to be the entire swarm's global best (gbest) position. Because all particles in the swarm are connected and each particle's pbest position is made known to all other particles, the topology in which the particles are connected follows that of the gbest neighbourhood described in (Kennedy et al. 2001).

**Particle Dynamics**

A conventional PSO algorithm (Kennedy and Eberhart 1995) with the velocity clamping rule introduced in (Blackwell and Bentley 2002a) is used in the Flying Geese recommender system. "Flying Geese" is thought to be a suitable name for this
recommender system as, in economics, it is a metaphor widely used to describe a situation where a more technologically advanced nation is the leader of the flock whom the other countries follow. This description also fits the particle dynamics for this work. One particle is selected to be the target (gbest) where other particles in the swarm follow. During a run, each particle moves towards both the current gbest and its own pbest position. The algorithm describing the Flying Geese PSO recommender system is shown below in figure 4.2.

```plaintext
LOOP 1 for each active user, A
  Create profile(A)
  Initialise swarm of particles, each with random initial position
LOOP 2 for each particle in swarm
  Set particle's pbest position to be its initial position
END 2 LOOP
LOOP 3
LOOP 4 for each particle in swarm
  Create an empty neighbourhood for A
  Map particle's current position to a set of feature weights, w
  LOOP 5 for each user j where j ≠ A and j is selected from current Profile Selection set
    Compute similarity value between A and j, euclidean(A,j) with w
    if similarity value is less than similarity threshold
      add j to A's neighbourhood
    END LOOP 5
  END LOOP 6
  Set fitness of current position to be the mean difference between predicted and actual ratings
  if this is the first iteration
    Set the swarm's gbest to be the pbest position that returns the best fitness score
  else
    LOOP 7 for each particle in the swarm
      if fitness of current position is better than fitness of particle's pbest
        Replace pbest position with current position
        if fitness of particle's pbest is better than fitness of the swarm's gbest
          Replace gbest position with this particle's pbest position
      END LOOP 7
  END LOOP 8
  Update the number of iterations by 1
  UNTIL the fitness score of gbest is below a threshold value or the maximum number of iterations is reached (END LOOP 3)
  Map gbest position to a set of feature weights and set as A's final feature weights
  Create an empty neighbourhood for A
  LOOP 9 for each user j where j ≠ A and j is selected from current Profile Selection set
    Compute similarity value between A and j using A's final feature weights
    if similarity value is less than similarity threshold
      add j to A's final neighbourhood
    END LOOP 9
  END LOOP 10 for each item i in test set for A
    Compute the predicted rating for i using A's final neighbourhood
    if the difference between predicted and actual ratings for i is 0
      Increment the number of correct predictions (zero tolerance)
    if the difference between predicted and actual ratings for i is either 1 or 0
      Increment the number of correct predictions (at-most-one tolerance)
END LOOP 10
END LOOP 1
```

Figure 4.2: The algorithm describing the Flying Geese PSO Recommender System

The particle dynamics are governed by the following three rules which update particle positions and velocities:
\[
V_i = wV_i + C_1P_1(X_{\text{best}_i} - x_i) + C_2P_2(X_{\text{gbest}} - x_i) \quad \text{Rule 1}
\]

if \(|v_i| > V_{\text{max}}\) \(v_i = \frac{V_{\text{max}}}{|v_i|}v_i\) \quad \text{Rule 2}

\[
x_i = x_i + v_i
\]

\text{Rule 3}

where:

- \(x_i\) is the current position of particle \(i\)
- \(x_{\text{best}_i}\) is the best position attained by particle \(i\)
- \(x_{\text{gbest}}\) is the swarm's global best position
- \(v_i\) is the velocity of particle \(i\)
- \(V_{\text{max}}\) is the maximum absolute value allowed for each component of \(v_i\) at each iteration
- \(w\) is a random inertia absolute weight between 0.5 and 1 (Eberhart and Shi 2001b)
- \(c_1\) and \(c_2\) are spring constants whose values are set to 1.494 (Eberhart and Shi 2001b)
- \(r_1\) and \(r_2\) are random numbers between 0 and 1 (Blackwell and Bentley 2002a)

As shown in figure 4.2, the only difference between the PSO and GA recommender systems is that lines 3 to 42 in figure 3.7 (which represented the genetic algorithm) are now replaced with lines 3 to 37 which describe the PSO algorithm.

**Experiments for Flying Geese PSO**

Four sets of experiments were conducted to observe the difference in performance between the PSO, GA (presented in chapter 3) and a standard, non-adaptive recommender systems based on the Pearson algorithm (Breese et al. 1998). In each set of experiments, the predicted ratings of all 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 ratings were then compared against those produced from the simple Pearson algorithm and the GA system.

The four sets of experiments were as follows:

**Experiment 1:** Each of the first 10 users (out of 944) was chosen as the active user, \(A\), in turn (loop1 starting at line 1 and ending at line 51 of figure 4.2). The first 10 users formed the Profile Selection set and were used to provide recommendations (loop 5 from line 11 to line 15 and loop 9 from line 39 to line 43 of figure 4.2).

**Experiment 2:** Each of the first 50 users was chosen as the active user, \(A\), in turn. The first 50 users formed the Profile Selection set and were used to provide recommendations. Note that the 10 users in experiment 1 were a subset of the 50 users here.
**Experiment 3:** Each of the first 10 users was chosen as the active user, $A$, in turn. For each run, 9 different users were selected randomly (out of 944) to form the Profile Selection set and were used to provide recommendations. All three systems used the same 9 randomly selected users at each run.

**Experiment 4:** Each of the first 50 users was chosen as the active user, $A$, in turn. For each run, 49 different users were selected randomly (out of 944) to form the Profile Selection set and were used to provide recommendations. All three systems used the same 49 randomly selected users at each run. Note that the 9 users in experiment 3 were a subset of the 49 users here.

These were performed on two versions of the systems:

**Zero tolerance** – the accuracy of the system is defined as the percentage of ratings which the system predicted that matched the actual rating of the active user.

**At-most-one tolerance** – same as zero tolerance but if the difference between the predicted and actual rating is less than or equal to 1 then this predicted rating is considered to be correct.

The results of these experiments were captured in loop 10 from lines 44 to 50 in figure 4.2. Table 4.1 below shows the parameters used in the experiments. Note that preliminary experiments were performed and the following set of parameter values was found to be most suitable. These values were kept the same in all four experiments (unless otherwise specified):

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Swarm size</strong></td>
<td>75</td>
</tr>
<tr>
<td>The number of particles in the swarm.</td>
<td></td>
</tr>
<tr>
<td><strong>Maximum number of iterations for each run</strong></td>
<td>300</td>
</tr>
<tr>
<td>If the number of iterations reaches this value and the solution has not been found, the best solution attained is used as the final result.</td>
<td></td>
</tr>
<tr>
<td><strong>Weight reduction size</strong></td>
<td>4</td>
</tr>
<tr>
<td>The scaling factor for the 18 movie genre frequencies.</td>
<td></td>
</tr>
<tr>
<td><strong>Number of runs</strong></td>
<td>30</td>
</tr>
<tr>
<td>The number of times the system was run for each active user, making 300 runs for experiments 1, 3 and 1500 runs for experiments 2, 4.</td>
<td></td>
</tr>
</tbody>
</table>
**Maximum velocity**
The maximum velocity for each particle during an iteration.

**Search space range**
The range in each dimension of the search space.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum velocity</td>
<td>20</td>
</tr>
<tr>
<td>Search space range</td>
<td>0-255</td>
</tr>
</tbody>
</table>

Table 4.1: Parameters values used in the experiments

**Results for Flying Geese PSO**
The results for experiments 1 to 4 with zero tolerance are shown in figures 4.3 to 4.6, respectively. Figures 4.7 to 4.10 display the results for experiments 1 to 4 with at-most-one tolerance. Finally, the average prediction accuracy for all experiments is shown in figure 4.11.

![Comparison between PA, GA and PSO - best results for experiment 1 with zero tolerance.](image1)

Figure 4.3: Comparison between PA, GA and PSO - best results for experiment 1 with zero tolerance.

![Comparison between PA, GA and PSO - best results for experiment 2 with zero tolerance.](image2)

Figure 4.4: Comparison between PA, GA and PSO - best results for experiment 2 with zero tolerance.
Figure 4.5: Comparison between PA, GA and PSO - best results for experiment 3 with zero tolerance.

Figure 4.6: Comparison between PA, GA and PSO - best results for experiment 4 with zero tolerance.

Figure 4.7: Comparison between PA, GA and PSO - best results for experiment 1 with at-most-one tolerance.
Figure 4.8: Comparison between PA, GA and PSO - best results for experiment 2 with at-most-one tolerance.

Figure 4.9: Comparison between PA, GA and PSO - best results for experiment 3 with at-most-one tolerance.

Figure 4.10: Comparison between PA, GA and PSO - best results for experiment 4 with at-most-one tolerance.
Whilst the predictions computed with the Pearson algorithm always remained the same given the same parameter values, those obtained from the PSO and GA varied 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 correct predictions) was chosen and plotted against the results from the Pearson algorithm. Again, 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 preferences of the active user.

By looking at the results obtained from the experiments, the following observations can be made:

- On the whole the PSO algorithm performed better than the other two systems, see figures 4.3 to 4.6 for results obtained from experiments 1 to 4. Figure 4.11 supports this claim.

- When random profile selection was used, in most cases the prediction accuracy for PSO rose and, did as well as, or outperformed the other two systems. Figures 4.5 and 4.6 show the results for experiments 3 and 4.

- The overall 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 4.11 shows the average prediction accuracy for all four experiments with both zero and at-most-one tolerance levels.

• The speed of adaptation of the PSO system was approximately 10% faster than that of the GA. For 10 users, a typical run by the GA and PSO systems took approximately 350 and 310 seconds respectively. Similarly, for 50 users, a typical run by the GA and PSO systems took approximately 8310 and 7920 seconds respectively. Nevertheless, compared to the PA system (which took approximately 3 seconds for a typical run of 10 users and 20 seconds for a typical run of 50 users), the speed of the PSO system is still significantly slower and needs to be improved upon in order for it to be used in an online system.

• 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 effective the system was at using the information from this duplicate user to give recommendations. The results demonstrated that the best run was obtained from the runs which contained 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.

The standard deviation values of the results obtained in experiments 1 to 4 for the PA, GA and PSO systems were computed in order to examine the degree of variation in the prediction accuracy, refer to Appendix A.

For experiments 1 and 2 with fixed users in all runs, the predictions obtained from the Pearson Algorithm always remained the same, thus the standard deviation for the PA was zero in all cases. In experiment 1, the standard deviation was less than 4% on 8 out of 10 active users for both GA and PSO systems. For experiment 2, the standard deviation was less than 4% for 45 and 40 out of 50 users for the GA and PSO, respectively. In both experiments 1 and 2, the standard deviation of zero was returned in a few cases – for GA and PSO, this usually means that the system was unable to find any users similar to the active user or that those similar users had not rated any items in the test set, thus the mean rating of the active user was used. As expected, when a greater variety of profiles was considered due to random sampling in experiments 3 and 4, the standard deviation for these cases no longer returned zero.
Overall, the standard deviation in most cases was still less than 4% for most active users – this shows that the systems were able give accurate predictions with acceptable levels of variation between runs.

In most cases, the standard deviations were found to be similar for the PA (experiments 3 and 4), GA and PSO systems where the difference was usually less than 1% and therefore, for reasons of space, will not be explicitly given for future experiments.

**Analysis of Results for Flying Geese PSO**

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 made in chapter 3 (as the number of users goes up, the probability of finding a better matched profile increases and hence, the accuracy of the predictions should also increase) still applies to the GA system where the average accuracy of the experiments conducted with 50 users is greater than that of the same experiments with 10 users, see figure 4.11. However, this is not the case for the PSO recommender. One explanation to this fall in the average accuracy level is that in experiment 1 with 10 users, those that were selected to be in the neighbourhood could be highly similar to the active user. As the number of users increased, more users were being considered and this could sometimes result in many less similar users being added to the neighbourhood and hence, lowered overall prediction accuracy, see figure 4.11. This problem will be examined in more detail and tackled in later work where we treat users themselves as particles in the swarm, see chapter 5.

Results for the PSO from experiments 3 and 4 confirmed the observation made in the previous work on the GA recommender 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 amongst 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 did not have any effect on the performance of PSO in experiment 4 when random sampling was used.

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. As stated previously, in the GA recommender, when more than 1 run for an active user produced the same best performance (highest number of ratings being predicted correctly), results indicate that the same set of users had been selected to be in the neighbourhood to give recommendations. However, this was not always the case for the PSO system. 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 was this position that other particles moved towards. It was 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. Later work on variations of PSO 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.

4.2. Variations of PSO Recommender

Based on the comparison in section 4.1 above, the Flying Geese (FG_PSO) algorithm is investigated further by examining its variations: Quad Feature, Migrating and Flocking. Four experiments were conducted for each variation to assess performance and the results obtained were compared against those of the original FG_PSO recommender. Note that the results with at-most-one tolerance are not shown separately in this section as they are similar to those obtained with zero-tolerance (zero-tolerance is a subset of at-most-one tolerance).

4.2.1. Variation 1: Quad Feature PSO (QF_PSO)

As shown previously, both GA and Flying Geese PSO recommenders outperformed the non-adaptive PA approach that uses only one feature, movie rating, to calculate the similarity between users. Flying Geese PSO defines the user’s recommendation preferences to be of 22 features: movie ratings, age, gender, occupation and 18 movie
genre frequencies which represent the number of times the user rated movies that belong to a given genre. To assess the impact of these features on the system performance in terms of prediction accuracy, a new system employing only the first four features is introduced. The new search space for this variation therefore becomes a 4-dimensional one.

Experiments 1 for Variation 1

The same four experiments were carried out to assess the performance of Quad Feature PSO. All the parameter values were kept the same.

Results 1 for Variation 1

The results obtained for Quad Feature PSO (QF_PSO) are shown below, showing the best of 30 runs. These are compared against those of the Flying Geese PSO (FG_PSO). The results for experiments 1 to 4 with zero tolerance are displayed in figures 4.12 to 4.15, respectively. The average prediction accuracy for all experiments is shown in figure 4.16.

![Graph](image)

Figure 4.12: Comparison between QF_PSO and FG_PSO - best results for experiment 1 with zero tolerance.
Figure 4.13: Comparison between QF_PSO and FG_PSO - best results for experiment 2 with zero tolerance.

Figure 4.14: Comparison between QF_PSO and FG_PSO - best results for experiment 3 with zero tolerance.

Figure 4.15: Comparison between QF_PSO and FG_PSO - best results for experiment 4 with zero tolerance.
Looking at the results in figures 4.12 to 4.15, the differences in prediction accuracy of these two systems were small, therefore it can be concluded that these two systems performed approximately the same, see figure 4.16 for average correct predictions for all experiments with zero and at-most-one tolerance levels. In addition, average and worst results were examined and compared amongst the two approaches. Again, the results were similar for the two modes considered.

As the two systems performed equally well, this implies that the 18 genre features that were not used in Quad Feature PSO have little impact on prediction accuracy. In some cases, this could be due to the fact that there were no similar users to the active user which meant that the mean rating of the user had to be used. This would therefore make the predicted rating the same for both systems.

Standard deviations for QF PSO were computed and found to be similar to those for FG PSO.

**Experiments 2 for Variation 1**

In Flying Geese PSO, the importance of the 18 genre frequencies was reduced by a given factor, the weight reduction size. This was carried out because the 18 genres could be considered to be 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. Following the earlier comparison between Quad Feature PSO and Flying Geese PSO, another reason for the results of these two systems being similar could be that the weight reduction size which
has always been set to 4 might have an impact on performance. Because of this, different weight reduction sizes were investigated in order to assess the importance of this setting. The same experiments, outlined in the Flying Geese section, were carried out with the weight reduction size set to 1 (no reduction) and 8 (larger reduction).

Results 2 for Variation 1
The results for different weight reduction sizes were compared against those obtained with the original size i.e. 4 used in the Flying Geese PSO. These are shown in figures 4.17 below.

It was found that results did not vary much. Additionally, it was seen that in the cases where the active user’s mean rating was used i.e. users 3, 4 and 6 in experiment 1, the results remained the same for different weight reduction sizes. This suggests that the weight reduction size does not have any impact on performance and hence confirms the earlier presumption that these 18 genre features play a minimal role in improving prediction accuracy of the PSO recommender system for the active users chosen.

4.2.2. Variation 2: Migrating PSO (M_PSO)
The update algorithm used in Flying Geese PSO was taken from the original PSO algorithm by Kennedy and Eberhart (1995). The rules stated that the \( p_{best} \) and \( g_{best} \) positions were updated only if the new position produced a higher fitness score. However, when a 3D visualisation tool showing the swarm movements was used (details of this are
given in section 4.3.1 of this chapter), it was discovered that many of the runs did not converge. The particles' pbest positions, in many cases, remained the same throughout the run. More precisely, these particles either returned the same fitness score as the gbest or they were still moving towards a better position i.e. the gbest when the maximum number of iterations was reached. When there were several positions that returned the same `best` fitness score, the swarm selected one of them at random to be its gbest. There can only be a single solution at any one time to represent the active user's current preferences and therefore the final gbest position is used for this. Nonetheless, this does not seem like the best approach. If a new position is encountered which returns the same fitness score as the particle's personal best then the pbest should be updated with this new position. It could be considered as a particle moving in the "right" direction towards a better solution. Moreover, the gbest should also be updated if other positions return the same fitness score so other equally good areas could be examined. The original update algorithm was therefore modified so that the pbest and gbest positions would be updated if the new position returns the same or a better fitness score (loop 7 starting at lines 24 and ending at line 28 in figure 4.2 were replaced by figure 4.18 below). The name, Migrating PSO, was thought to be suitable for this variation of the algorithm as it describes the behaviour of the swarm; particles are regularly moving from one position to another and eventually settling in one place.

```plaintext
24 LOOP 7 for each particle in the swarm
25 if fitness of current position is the same as or better than fitness of particle’s pbest
26 Replace pbest position with current position
27 if fitness of particle’s pbest is the same as or better than fitness of the swarm’s gbest
28 Replace gbest position with this particle’s pbest position
29 END LOOP 7
```

Figure 4.18: Modified pbest and gbest update method for M_PSO (replace loop 7 in figure 4.2)

Experiments for Variation 2

Experiments outlined in the Flying Geese section were performed in order to compare the performance of Migrating and Flying Geese PSO. They were conducted under controlled initial positions whereby both systems used the same set of initial random positions for each run. The number of times the system was run for each active user was changed to 5, making 50 runs for experiments 1, 3 (with 10 users) and 250 runs for experiments 2, 4 (with 50 users).
Results for Variation 2

Figures 4.19 to 4.22 below display the results obtained from M_PSO and FG_PSO with zero tolerance. The average prediction accuracy for all experiments is also shown in figure 4.23.

![Comparison between FG_PSO and M_PSO - best results for experiment 1 with zero tolerance.](image1)

![Comparison between FG_PSO and M_PSO - best results for experiment 2 with zero tolerance.](image2)

![Comparison between FG_PSO and M_PSO - best results for experiment 3 with zero tolerance.](image3)
As shown above, the results obtained were similar for both systems. However, by using the 3D visualisation tool to display the movements of the particles’ $p_{best}$ positions, it was seen in most cases for Migrating PSO that the particles rapidly converged into a single solution. This meant that the final solution was one that all or most particles ‘agreed’ on. Figure 4.24 below shows the snapshots of the particles’ $p_{best}$ positions at iteration 300 for the two systems. The black circle represents the final $g_{best}$ position i.e. final solution for this run.
It was not surprising that the time the swarm took to achieve the solution was much less for Migrating PSO. Assuming the paths that the particles took towards the gbest often returned the same or better fitness score, this means that the pbest positions were constantly updated and therefore the distances that the particles had to explore between their pbest and gbest always shortened with increasing iterations. Even though Migrating PSO is a fast method for achieving a single solution, there is a high risk that the swarm may experience premature convergence on local minima and therefore, the final solution might not be the best possible one.

Similar standard deviations were returned by both FG PSO and M PSO. However, in M PSO, there were a few users in experiment 3 whose standard deviation returned zero, which implies that there were no similar users to these active users. Looking at the prediction accuracy, it was found that FG PSO performed better on these users. This therefore demonstrates that there were similar users for these active users and thus, M PSO prematurely converged.

4.2.3. Variation 3: Flocking PSO (F PSO)

As stated previously, the problem of capturing the user’s preferences is not trivial. More than one set of feature weights (positions) can result in the same fitness score. This is partly due to the rounding effect; predicted ratings being rounded to the nearest integer between 1 and 5. Because of this, many runs from the Flying Geese PSO did not converge. As shown earlier in figure 4.24, in these runs, the particles’ pbest positions remained scattered throughout the space and did not swarm around the gbest position. The Migrating PSO attempted to overcome this problem but a risk of premature convergence was discovered. Flocking PSO is another variation of Flying Geese PSO which attempts to overcome the problem of multiple solutions. In order to reduce the number of premature convergence cases, it was decided that the original pbest update (lines 25 and 26 in figure 4.2) should be kept, i.e. update only if the new position is better. The main difference here lies in the gbest update. When there are several positions that return the same best fitness score, a central position of all these “best” positions is set to be the swarm’s gbest (loop 7 in figure 4.2 is now replaced by figure 4.25 below). This allows all the ‘best’ positions an equal chance of being explored. At every iteration, the gbest position is required to be recalculated even if its fitness score remained the same as the previous. This is because more particles may have discovered new positions that also
return the same score as that of the \textit{gbest} and hence a new central position. The name, Flocking PSO, was chosen because the new \textit{gbest} calculation resembles the Flock Centering rule from the famous Flock simulation algorithm by Craig Reynolds (1987).

\begin{verbatim}
24 LOOP 7 for each particle in the swarm
25 if fitness of current position is better than fitness of particle's pbest
26 Replace pbest position with current position
27 if fitness of particle's pbest is better than fitness of the swarm's gbest
28 Replace gbest position with this particle's pbest position
29 else if fitness of particle's pbest is the same as fitness of the swarm's gbest
30 Replace gbest position with central position of particles that return this fitness so far
31 END LOOP 7
\end{verbatim}

Figure 4.25: Modified \textit{gbest} update method for Flocking PSO (replace loop 7 in figure 4.2)

Experiments for Variation 3

Experiments outlined in Flying Geese section were conducted with fixed initial random positions. Both systems were performed 5 times for each active user, making 50 runs for experiments 1, 3 and 250 runs for experiments 2, 4.

Results for Variation 3

The results obtained for experiments 1 to 4 with zero tolerance are shown in figures 4.26 to 4.29, respectively. The average prediction accuracy for all experiments is shown in figure 4.30.

![Graph showing comparison between FG PSO and F PSO](image_url)

Figure 4.26: Comparison between FG PSO and F PSO - best results for experiment 1 with zero tolerance.
Figure 4.27: Comparison between FG_PSO and F_PSO - best results for experiment 2 with zero tolerance.

Figure 4.28: Comparison between FG_PSO and F_PSO - best results for experiment 3 with zero tolerance.

Figure 4.29: Comparison between FG_PSO and F_PSO - best results for experiment 4 with zero tolerance.
Again, the difference in prediction accuracy of the two approaches is relatively small. However, standard deviations were computed and it was found that the standard deviation for most active users was slightly smaller for F_PSO than that for FG_PSO. This shows that the predictions produced by F_PSO were more consistent than those by FG_PSO. However, in most cases where both systems converged, the time taken to attain the final solutions was noticeably less for Flocking PSO. Figure 4.31 below shows the movement of the gbest position for the two systems in a run. The iteration where the movement first reached zero and remained the same until the end of the run was taken to be the time that the system took to attain the final solution. It is clearly seen in the example below that Flocking PSO was faster in attaining the solution, reaching zero by iteration 40, compared to 70 for Flying Geese PSO. This result was consistently seen in 76.1% of the runs which converged.
Furthermore, by using the 3D visualisation tool to study the swarm movements, it was noticed that in many runs for Flocking PSO that did not converge, the particles formed an area of good solutions. The particles located within this area actually returned the same best score as the gbest, i.e. multiple solutions. As described earlier, Flocking PSO was created in an attempt to deal with this situation by constantly recalculating the gbest position to take into account all best solutions. Figure 4.32 below illustrates the gbest selection process for the two systems, the grey area shows an area of good solutions, the white square represents the current gbest position, black circles are particles that return the same fitness score as the current gbest and finally white circles show other particles in the swarm.

![Figure 4.32: The gbest selection process for Flocking and Flying Geese PSO](image)

In Flocking PSO (see left branch of figure 4.32), the white particles at iteration $i$, would move towards the gbest in the direction of the arrow. By iteration $i+1$, two additional particles would fall into the good area and the gbest is updated accordingly. Note that a distance that a particle can travel per iteration is limited by the maximum velocity ($v_{max}$), specified in table 4.1. Because of the constant update of gbest, it is not surprising that
Flocking PSO is faster in converging to either a single solution or a good area. More precisely, in the extreme cases for Flying Geese PSO like the one illustrated above where the particle selected to be \textit{gbest} is on one side of the search space. It would take longer for other particles on the other side to reach that position. Moreover, by not neglecting any of the ‘best’ particles, Flocking PSO can ensure that in most cases all potentially good areas are explored and hence, the solution attained is the best possible for the run.

4.3. Analysis of Flocking PSO

Previous experiments have indicated that the Flocking PSO is perhaps the most suitable of the PSO algorithms tested in the recommender system. It converges faster and takes into account all potentially good areas. This section further analyses all results obtained for Flocking PSO in order to understand and assess its utility within the recommender system. From careful study of swarm dynamics in Flocking PSO, six behaviours were most commonly observed. Note that the swarm is capable of other behaviours for other problems.

4.3.1. Software Analysis Tools

Huge amounts of data were collected during the experiments. For each run, the position of each particle at every iteration was recorded, together with its \textit{pbest} position; these were required in the reconstruction of movements by the 3D visualisation tool, described below. The swarm’s \textit{gbest} positions and their fitness score were also vital in determining how the final solution was reached. Over 7 Gigabytes of data were yielded from the conducted four experiments with the total of 3,600 runs. This would certainly take a very long time to be analysed by hand. Because of this, various software tools were created to aid the analysis process. These are as follows:

Swarm’s Prediction Accuracy Tool

The prediction accuracy tool works in the same way as the fitness function; given a position in the search space, the tool converts it into a set of weights and outputs the fitness score by calculating the number of predicted ratings that it computes correctly for both training and test movie items. This is performed for the final \textit{pbest} position of all particles in the swarm. The results will show how well the other particles did in
comparison to the final solution (\(g_{best}\)) obtained by the swarm. This is illustrated in figure 4.33 below.

![Diagram showing swarm's prediction accuracy tool](image)

**Figure 4.33: Swarm’s Prediction Accuracy Tool**

**Graphical Data Display Tool**

This contains many display options: average \(p_{best}\) movement, swarm activity, \(g_{best}\) fitness score and the final solution. For each option, the tool extracts appropriate portions of the data and displays them in a graphical format that is easier to comprehend. Detailed descriptions of these display options and their examples are shown later in the Swarm Diagrams section.

**3D Visualisation Tool**

This is a powerful tool used to reconstruct the paths of the swarm in any given run. Furthermore, the movements of the particles’ \(p_{best}\) positions throughout a run can also be recreated using this tool. Again, detailed descriptions of the two options are discussed below in the Swarm Diagrams section.

**Swarm Diagrams**

The following diagrams are examples of outputs that were created by the graphical data display and 3D visualisation tools. They are arranged in the same format which is used later in section 4.3.2 to describe the swarm behaviours.
Average $pbest$ Movement

At each iteration $i$, the difference between a particle’s current and its previous $pbest$ positions is calculated for all particles in the swarm. These differences are then summed up to give the total $pbest$ movement. This is then divided by the number of particles in the swarm to give the average movement for that iteration, see equation below. The diagram shows average $pbest$ movement for a run of 300 iterations i.e. $i=1$ to $i=300$.

$$avg\_movement = \frac{1}{z} \sum_{p=1}^{z} \left( \frac{1}{z} \sum_{j=1}^{z} |x_{pbest(i,f)} - x_{pbest(i-1,f)}| \right)$$  \[6\]
where:

- \( j \) is the number of particles
- \( p \) is the particle \( p \)
- \( x_{\text{pbest}p(i)} \) is the pbest position of particle \( p \) in the \( j \)th dimension at iteration \( i \)
- \( z \) is the number of features – in this case, 22

**Swarm Activity**

The calculation of the average swarm activity is similar to that of the average pbest movement above, see equation below. Instead of pbest positions, the current positions of particles are used. The diagram shows the level of swarm movement at each iteration \( i \), displaying the run’s robustness and convergence. For example, when activity\( i = 0 \), there is no movement and the swarm has converged to a single position.

\[
\text{activity}\, i = \frac{1}{j} \sum_{p=1}^{j} \left( \frac{1}{z} \sum_{f=1}^{z} \left| x_p(i, f) - x_p(i - 1, f) \right| \right) [7]
\]

where:

- \( j \) is the number of particles
- \( p \) is the particle \( p \)
- \( z \) is the number of features/dimensions
- \( x_{p,0,i} \) is the position of particle \( p \) in the \( j \)th dimension at iteration \( i \)
- \( z \) is the number of features – in this case, 22

**gbest Fitness Score**

The diagram displays the fitness of the swarm’s global best position at each iteration \( i \). This is a minimisation problem which means that a lower fitness score represents a better solution.

**Final Solution**

The final gbest position attained by the swarm in the given run is converted into a set of 22 feature weights that represents the active user’s recommendation preferences. The diagram displays these weights attained by the swarm for a given user in a run.
3D visualisation tool - pbest movements

The tool is capable of reconstructing the movements in 3D, showing the manner in which the pbest positions were updated throughout a run. However, the third dimension is not displayed as its visualisation is less clear. In this thesis, unless otherwise specified, the x and y axes default to the first two features; rating and age (as these have been shown to be most significant). The diagram displayed shows a screenshot of the particles’ pbest positions (light grey squares) at the last iteration (in this case, iteration 300) of a run. Each particle is labelled with its unique id, making the movements easier to track. The black circle shows the swarm’s gbest position for that iteration.

3D visualisation tool - swarm path

This is similar to the pbest movement reconstruction. Again, the third dimension is not displayed and x and y axes default to the first two features; rating and age. However, in this case the actual paths that the particles took in relations to the gbest position are shown. The diagram shows a screenshot of the final positions of all particles at iteration 300. Each position is denoted by a dark grey square and again labelled with the particle’s unique id. The black circle shows the swarm’s gbest position. By running this together with the pbest movement display, the swarm dynamics can be observed i.e. the particles move towards both their pbest and the swarm’s gbest positions whilst the gbest constantly adjusts its position to stay in the centre of all current best positions.

4.3.2. Swarm Behaviours

From the extensive study of the swarm dynamics together with the aid of the analysis tools described above, the following 6 distinct swarm behaviours were identified from the results obtained for Flocking PSO in section 4.2.3. By classifying the behaviour of the swarm into one of these behaviour categories, the nature of the final solution attained by the swarm can be understood i.e. whether this is the best possible solution. More importantly, this provides a good guidance in choosing the best solution to use out of those obtained from all runs for the final recommendation process. Furthermore, by knowing the nature of the solution, possible actions can be taken to improve its accuracy level further.
**Behaviour 1 – No Convergence with Multiple Solutions**

This behaviour typically represents a run in which either there is more than one possible solution or the particles have not yet discovered a better position i.e. the $g_{best}$ position. To investigate this claim, the Swarm’s Prediction Accuracy tool was performed on all $p_{best}$ positions on the training items for more than 50 runs which belong to this category. The results showed that for the $p_{best}$ positions surrounding the $g_{best}$, their fitness was the same as that of the $g_{best}$ (otherwise, these $p_{best}$ positions would have been updated by the $g_{best}$). Moreover, the $p_{best}$ positions that were not close to the $g_{best}$ returned worse fitness score than that of the $g_{best}$, as was expected.

---

![Average pbest Movement](image1.png)

![Swarm Activity](image2.png)

![gbest Fitness Score](image3.png)

![Final Solution](image4.png)
Average pbest movement  
a series of sudden movements spread over time with decreasing magnitude.

Swarm Activity  
small oscillating movement with a slight downward trend.

gbest fitness score  
either no change in the gbest fitness, or changes earlier on followed by no more change.

Final Solution  
feature weights representing the active user’s preference are satisfactory but still require more fine-tuning.

pbest Movements  
most pbest positions are located around the gbest forming an area of equally good solutions and the rest are scattered around the search space.

Swarm Path  
there are still particles whose pbest positions are scattered around the space and not converging towards gbest as they return the same fitness score. This results in these particles oscillating between pbest and gbest.

**Behaviour 2 – No Solution**

This behaviour is commonly found in a run where no ‘best’ solution can be found to represent the active user’s preference. Again, the prediction accuracy tool was used and it was found that almost all positions in the search space returned the same fitness score, hence no movement in pbest. The gbest therefore remained in the centre of all particles in the swarm, which was usually the middle of the swarming space. For this reason, the final feature weights for the runs in this category always looked the same, i.e. high emphasis on the first four features.
## Behaviour Feature Characteristics

<table>
<thead>
<tr>
<th>Behaviour Feature</th>
<th>Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average $p_{best}$ movement</td>
<td>no movement because all positions return the same fitness score and the mean rating is being used.</td>
</tr>
<tr>
<td>Swarm Activity</td>
<td>high levels of oscillating movement throughout the run.</td>
</tr>
<tr>
<td>$g_{best}$ fitness score</td>
<td>no change in fitness.</td>
</tr>
</tbody>
</table>
Final Solution equally high emphasis on the first four features with an equally low weight for the rest of the features. This means that all the features carry the same weights but due to the weight reduction size for the 18 genre features, the first four features are given higher importance.

*pbest* Movements the *pbest* positions are scattered around the space. They remain in their initial position and do not get updated.

Swarm Path the particles oscillate between the *gbest* position (which always stays in the centre of all particles as all particles are equally fit) and their *pbest* position.

**Behaviour 3 – Good Convergence with Best Solution**

This behaviour shows a typical best or good run in which a single solution is found by the swarm.

<table>
<thead>
<tr>
<th>Average pbest Movement</th>
<th>Swarm Activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>gbest Fitness Score</td>
<td>Final Solution</td>
</tr>
</tbody>
</table>

137
Figure 4.37: Behaviour 3 example

<table>
<thead>
<tr>
<th>Behaviour Feature</th>
<th>Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average <em>pbest</em> movement</td>
<td>large movement at earlier iterations which dampen with increasing iterations with perhaps, a few ‘spikes’ at intervals. The <em>pbest</em> positions stabilise i.e. no movement at later iterations. Again, it was noticed that an increase in the movement corresponds to a change in the <em>gbest</em> position.</td>
</tr>
<tr>
<td>Swarm Activity</td>
<td>huge movements at earlier iterations due to large distance that the particles have to move between their <em>pbest</em> and the swarm’s <em>gbest</em> positions. The intensity of the swarm activity reduces gradually to small oscillations then stops completely as the swarm converges.</td>
</tr>
<tr>
<td><em>gbest</em> fitness score</td>
<td>reduction in the fitness score.</td>
</tr>
<tr>
<td>Final Solution</td>
<td>Best feature weights that describe the user’s preferences. Clear emphasis on features that are important to the active user. The weights for the rest of the features are usually zero.</td>
</tr>
<tr>
<td><em>pbest</em> Movements</td>
<td><em>pbest</em> positions gradually converge into one position.</td>
</tr>
<tr>
<td>Swarm Path</td>
<td>the particles move to one single position. In this example, the convergence occurred just after 160 iterations when both the average <em>pbest</em> movement and the swarm activity are equal to zero.</td>
</tr>
</tbody>
</table>

**Behaviour 4 – Premature Convergence with Undesirable Solution**

This behaviour shows a run with premature convergence. It is particularly difficult to determine as the characteristics of the behaviour can be similar to those described for behaviour 3. The runs belonging to this category usually converge very quickly without
exploring other areas in the search space. Thus, this is not the best run for the active user. Furthermore, examining the results for this active user would show that better results are produced by other runs.

Figure 4.38: Behaviour 4 example
<table>
<thead>
<tr>
<th>Behaviour Feature</th>
<th>Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average pbest movement</td>
<td>very large movement at earlier iterations. The movement then reduces to zero at a fast rate and stabilises at zero for the rest of the run.</td>
</tr>
<tr>
<td>Swarm Activity</td>
<td>inverse exponential drop reaching zero movement earlier on. The run starts with very high intensity of swarm activity.</td>
</tr>
<tr>
<td>gbest fitness score</td>
<td>a huge drop in fitness at earlier iterations before stabilising at this score throughout the run.</td>
</tr>
<tr>
<td>Final Solution</td>
<td>usually not the best feature weights</td>
</tr>
<tr>
<td>pbest Movements</td>
<td>all pbest positions converge into one position very quickly</td>
</tr>
<tr>
<td>Swarm Path</td>
<td>the particles rapidly move to a single position without exploring any other areas in the space i.e. fall into a local minima.</td>
</tr>
</tbody>
</table>

**Behaviour 5 – Slow Convergence with Good Solution**

This behaviour shows a scenario where the swarm finds an area of good solutions and is perhaps en route to finding the best single solution. Final solutions from runs belonging to this category are usually either the best or very good.

![Average pbest Movement](image1)

![Swarm Activity](image2)

140
### Behaviour Feature

<table>
<thead>
<tr>
<th>Behaviour Feature</th>
<th>Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average pbest movement</td>
<td>oscillating movement with high intensity at the start. This could have been mistaken to be the 4th behaviour above. However, the difference between this and the premature convergence behaviour is that the movement does not stabilise at zero. In some cases, the movement oscillates near zero for a period of time before a change in the gbest position causes the average pbest movement to increase again.</td>
</tr>
<tr>
<td>Swarm Activity</td>
<td>mimics the movements of pbest. In some cases, the movement gradually comes down as if it were to converge then a change in the gbest again causes the intensity of swarm activity to pick up.</td>
</tr>
<tr>
<td>gbest fitness score</td>
<td>usually small reduction of fitness occurring throughout the run.</td>
</tr>
<tr>
<td>Final Solution</td>
<td>good feature weights – still require additional fine-tuning</td>
</tr>
<tr>
<td>pbest Movements</td>
<td>the pbest positions are gradually updated to be around the gbest.</td>
</tr>
<tr>
<td>Swarm Path</td>
<td>the particles swarm into an area around the gbest position when the average pbest movement and swarm activity are near to zero</td>
</tr>
</tbody>
</table>

---

**Figure 4.39: Behaviour 5 example**

- gbest Fitness Score
- Final Solution
- pbest Movements
- Swarm Path
for the first time. If the movement later goes up due to a change in the gbest position, all particles also move towards the new gbest position as a group.

**Behaviour 6 – Convergence to Good Area**

This behaviour shows a scenario where the swarm finds an area of equally good solutions. The runs associated with this behaviour are normally the best or very good.
The six behaviours described above were also true for Flying Geese PSO, except for the second behaviour (No Solution). In Flocking PSO when there was no single solution, the final set of weights attained by the swarm was always consistent i.e. the gbest position was in the middle of the search space. However, given the same situation, the set of weights attained from the Flying Geese PSO approach was inconsistent. This is due to the fact that only one out of many ‘best’ particles was randomly selected to be the gbest.

The behaviours for Migrating PSO were different from those of Flocking PSO. Most runs in Migrating PSO consisted of movement starting off high and rapidly reducing to stationary. The swarm activity diagram for this approach would look similar to either
behaviour 3 or 4 of Flocking PSO where convergence occurred. An example of average *pbest* movement and swarm activity for a run from Migrating PSO is shown in figure 4.41.

Figures 4.42 to 4.45 show the distribution of the six behaviours for active users 1 to 10 in all 4 experiments for Flocking PSO. The total number of runs for each experiment is 50 (5 runs per user).

✓✓✓ = Extremely desirable

××× = Extremely undesirable

---

Figure 4.41: Example movement of Migrating PSO

Figure 4.42: Distribution of behaviours for experiment 1
As shown above, it was interesting to see that the first two behaviours were more common in experiments 1 and 3 where the number of users was 10 (instead of 50 for the other 2 experiments). Moreover, the second behaviour (no solution) occurred the most in
these two experiments which was not surprising as the smaller the number of users, the less chances of finding similar users. The results for experiments 2 and 4 seemed to corroborate this view as none of the runs belonged to behaviour 2. Furthermore, the last 2 behaviours, in particular behaviour 6, occurred more often in experiments 2 and 4. Again, because the number of users was higher for these two experiments, the chance of good solutions being found was also higher. This was confirmed as behaviour 3 (most desirable) which is good convergence with best solutions also occurred more in experiments 2 and 4.

4.3.3. A Variation of Flocking PSO

From the analysis, one final point that can be addressed is possible improvements for the flocking PSO algorithm. This section explores a variation of the flocking PSO recommender system to improve the performance of the system, especially runs in which behaviours 1 or 5 occur. By increasing the maximum velocity which affects the second rule of the update algorithm in section 4.1.1, the distance that particles are allowed to travel per iteration is increased. The runs that belonged to behaviour 1 did not converge because many particles did not reach the gbest position due to limited number of iterations allowed per run. With increased maximum velocity, this problem is reduced as particles can travel a greater distance and therefore the gbest can be reached faster. Again, with increased maximum velocity allowed, the problem found in behaviour 5 of slow convergence can also be reduced.

Experiment for a variation of Flocking PSO

Experiments outlined inFlying Geese section were conducted on the Flocking PSO (F_PSO) with maximum velocity set to 255 (F_PSO_255). This gives the particles the capability to travel from one side of the search space to the other in a single iteration, if required.

Results for a variation of Flocking PSO

The results obtained for experiments 1 to 4 with zero tolerance are shown in figures 4.46 to 4.49, respectively. The average prediction accuracy for all experiments is shown in figure 4.50.
Figure 4.46: Comparison between 2 different maximum velocities in F_PSO - best results for experiment 1 with zero tolerance.

Figure 4.47: Comparison between 2 different maximum velocities in F_PSO - best results for experiment 2 with zero tolerance.

Figure 4.48: Comparison between 2 different maximum velocities in F_PSO - best results for experiment 3 with zero tolerance.
Although subtle, from the results, it can be seen that in many cases the variation of Flocking PSO with increased maximum velocity to 255 performed better than the original Flocking PSO. More precisely, in experiments 1 and 3 where many runs in the original Flocking PSO fell under behaviour 1, see figures 4.42 and 4.44. In order to investigate this further, a similar distribution chart of the six behaviours for the variation of Flocking PSO was created and this is shown in figures 4.51 to 4.54. As expected, the number of runs that was categorised for the first behaviour is greatly reduced with the modified Flocking PSO. Furthermore, the number of runs which belonged to behaviour 3 also increased with the new system. The two systems performed equally well in experiment 4. This was because most runs in this experiment already fell under desirable behaviours in the original Flocking PSO approach. A drawback to this variation is that allowing the
particles to move a greater distance can increase the chance of premature convergence, as also shown in this analysis – see figures 4.51, 4.52 and 4.54.

Figure 4.51: Distribution of behaviours for experiment 1

Figure 4.52: Distribution of behaviours for experiment 2

Figure 4.53: Distribution of behaviours for experiment 3
Another action that can be taken to improve the prediction accuracy is to allow the swarm to run for longer, particularly the runs that fall under behaviour 5 (Slow Convergence) where the swarm activity is still high.

The final observation from the analysis is that it was not always the case that the solution which achieved the best fitness during training was the best at predicting ratings for test or new items. This could be due to the problem of overfitting i.e. the solution was adapted too specifically to the training items. By examining the way in which the solution was reached, it is possible to predict the reliability of the results.

As previously mentioned, because ratings are subjective (i.e. dependent on the time of day or the mood of the user at the time), noise in data can be introduced i.e. a rating given on an item may have been different if rated at a different time. Because of this, the system can sometimes be misled by the noise and hence, the solution attained does not represent the active user correctly. However, in the real world scenario users themselves continuously give feedback on the predicted ratings given by the system; they are asked to provide their rating on the items recommended and the system treats these ratings as new data to fine-tune the existing user preferences. This would therefore help reduce noise in data.
4.4. Summary

This chapter has shown how particle swarm optimisation 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 Flying Geese PSO system outperformed a non-adaptive approach and obtained higher prediction accuracy than the genetic algorithm system in most cases. In addition, compared to the GA approach, the PSO algorithm achieved the final solution faster, making it a more efficient way of improving performance where computational speed plays an important part in recommender systems. Nonetheless, it was discovered that in the PSO system, as more users were considered, this sometimes resulted in many less similar users being added to the neighbourhood and hence, lowered the overall prediction accuracy.

Three variations of the original Flying Geese PSO algorithm were presented. Firstly, the Quad PSO using only the first four features was examined and it was found that the difference in performance between this and the Flying Geese systems was negligible. For the active users chosen, there was no clear difference in performance between 18 and 4 genres. Secondly, the Migrating PSO was considered. In this variation, the rule for updating the particle’s pbest position and the swarm’s gbest was modified in an attempt to obtain convergence in most runs. Even though, the results were again similar for both Migrating and Flying Geese systems, it was discovered that there was a higher risk of premature convergence in Migrating PSO. The last variation, the Flocking PSO, was introduced to deal with multiple solutions and reduce the number of cases with premature convergence. This was shown to be a better algorithm for providing recommendations.

Various software tools were developed to analyse the Flocking PSO algorithm in more detail. As a result, 6 distinct swarm behaviours were identified. This has shown to be a useful way of determining the reliability of the solutions attained by the swarm. Finally, improvements to the Flocking PSO Recommender were discussed and its variation was presented which outperformed the original one.
CHAPTER 5

ClusterPSO: A Novel Swarming Algorithm

It was successfully shown in chapters 3 and 4 that a system which employs an adaptive technique (either GA or PSO) is able to make more accurate predictions, by fine-tuning neighbourhood selection, than one using a non-adaptive approach, i.e. Pearson algorithm (PA). This finding reinforces the fact that the success of recommender systems based on a collaborative filtering method lies in the neighbourhood selection process.

In chapter 3, it was discovered that as the number of users increased, the accuracy of the predictions by the GA recommender also improved. This was, however, not the case for the PSO recommender system in chapter 4. It was found that when more users were being considered, this sometimes resulted in many less similar users being added to the neighbourhood, and hence, lowered the overall prediction accuracy. Nonetheless, the comparison between the three systems found that the PSO approach was still able to outperform the GA and PA systems in most cases. The speed of adaptation (the time taken to attain the final set of feature weights for an active user) was also observed to be faster for the PSO than for the GA system.

Clearly, PSO is a better approach in terms of prediction accuracy and speed but further investigation into the neighbourhood selection process is required. This chapter therefore introduces a novel recommender system, ClusterPSO, that employs dynamics similar to those found in the PSO system to mimic real-world “friend formation” scenarios. This allows the users to dynamically select their own neighbourhood of ‘friends’ and avoid those that are dissimilar to them.

The chapter is organised as follows: section 5.1 describes the novel ClusterPSO recommender system. Section 5.2 provides preliminary cluster analysis using manual cluster allocation and an existing clustering algorithm, COP_KMEANS. Section 5.3 provides experimental results and analysis, together with a comparison of the ClusterPSO system against the PA, GA and PSO systems. Finally, section 5.4 concludes.
5.1. ClusterPSO Recommender

Both GA and PSO recommender systems were based on collaborative filtering, whereby a set of users similar to the active user is found and used to predict the active user's rating for a given item. The ClusterPSO recommender in this chapter follows once again the same idea of using similar people to give recommendations but the method used is rather different. So far, this process of neighbourhood selection in the GA and PSO approaches has been performed sequentially for each active user. The feature weights representing the recommendation preferences of a user is either evolved by the GA or attained by the PSO for one active user at a time. However, in this chapter, the concept of feature weights is temporarily removed and we focus on neighbourhood selection by ClusterPSO. The ClusterPSO system allows the users to run in parallel, moving towards those that are similar whilst avoiding those dissimilar to themselves. This results in a dynamic clustering of similar users. This way the neighbourhood selection can occur for all users simultaneously and thus, the speed in which recommendations can be provided is greatly improved. The dynamics in which the users move are governed by a set of modified PSO rules which are described later.

5.1.1. Swarming Space

In ClusterPSO, the word "user" is now used to mean particle, since each particle represents a specific user profile. The swarming space is defined as the space within which the users move. In ClusterPSO, the position of a user is represented as a point in an arbitrary 2-dimensional space at which the movement of the neighbourhood selection occurs.

Bounded Space

From earlier experiments with 10 users using a bounded swarming space, use of the visualisation tool showed that users often ended up stranded in the four corners of the space, see figure 5.1.
This was due to the repulsive forces (described later) that exist between dissimilar users whose purpose is to separate them. Since the swarming space is bounded, users at the corners have limited movement and often remain there. This highlights the need for an unbounded space where users can move from one side of the space to the other.

**Unbounded Space**

The swarming space used in this chapter has no boundaries. The shape of the swarming space can be described as being similar to a donut and, when displayed in 2 dimensions, it resembles a rectangle or a square (depending on the width and length) where parallel edges meet each other, see figures 5.2 and 5.3.
5.1.2. Users as Particles

The core part of the algorithm, the neighbourhood selection, is inspired by the real world scenario: formation of friends. For example, when a person first joins a new organisation such as a school or a workplace, he initially does not know anyone. By meeting new people and getting to know them, he can form an opinion about each of them and vice versa. In most cases, if two people are demographically similar (e.g. in the same age group) and/or have similar taste (e.g. in music or movies) then they are more likely to be attracted to each other and become friends. On the other hand, if two people do not share any common attributes, they are more likely to have fewer interactions between each other. This idea of friends forming naturally matches with the objective of the profile matching task in recommender systems where similar users have to be found.

In this system, only the first four features are considered in the user profile; they are the ones that were found in chapter 4 (Quad Feature PSO) to be most effective in achieving good results. These features are movie rating, age, gender and occupation. As mentioned earlier, for this chapter, it is assumed that these four features contribute equally and thus, a personalised set of feature weights for each active user is not required.

The novel idea of this work lies in the way that users themselves are used as particles in the algorithm. They continuously move around the swarming space, meeting other users and updating their list of friends (those that are similar to them). Initially, when a user
first joins the system, his profile is created by the profile generator (described in section 3.2.1). He is then assigned a randomly generated position in the space where he can immediately start to ‘mingle’.

5.1.3. Neighbourhood of Friends

As in the real life, most people have a group of friends whose advice can have some influence on their decision making. These friends are the ones that can make suggestions based on their relationship and familiarity. This work uses the term, neighbourhood of friends, to mean a group of users that are used to provide final recommendations to the active user. Each active user is responsible for finding his neighbourhood of friends.

This work introduces the notions of ‘best friends’ and ‘outsiders’. These two terms are described as follows.

**Best Friends**

Again, most people have a friend whose opinion they value the most out of the whole group. In ClusterPSO, the term, best friend, is used to represent a user in the neighbourhood that is most similar (with highest similarity value) to the active user. It is this friend that the active user remembers throughout the run. This concept is similar to the \( p_{best} \) (personal best) location used in the PSO system. However, the difference is that the \( p_{best} \) location is static, whereas here the best friend himself remains the same but his location constantly changes as he also moves around the space trying to find his own neighbourhood of friends.

Note that best friends are not symmetric; it is not necessarily true that if \( x \) is \( y \)'s best friend, then \( y \) is also \( x \)'s best friend. However, that is usually the case. Moreover, given the scalability of the system (a large number of users) and the simplified similarity measure with 4 features, it is possible that several users may have the same similarity value and thus, it is possible for a user to have more than one best friend. If this is true, the central location of all best friends' current position is used.
Outsiders

Another novel idea introduced in this work is the notion of outsiders. These are users that appear in the neighbourhood but do not belong there. Outsiders are considered dissimilar to the active user at this point in time as they do not share any common movies. The similarity value (returned by the Euclidean similarity measure described later) is set to a reserved value of -1 in order to mark these outsiders for removal from the active user's neighbourhood (see lines 14 and 56 in figure 5.11).

When the active user encounters an outsider, he adds this user to his outsider list. The list is used to help the active user move in a direction opposite to these outsiders, and hence reduce the number of outsiders in the neighbourhood. The list is reinitialised at the beginning of each iteration to allow the possibility that, at future encounters, these previous outsiders may have rated new movies that are similar to the active user and hence could develop into a friend.

5.1.4. Neighbourhood Selection

In this work, the neighbourhood selection is different than the one used in the GA and PSO systems. As mentioned earlier, here the users themselves are responsible for exploring the space, finding their neighbourhood of friends. This process occurs simultaneously for all the users, each acting as an active user in his own right. Neighbourhood selection is not a trivial task, consisting of 5 sub-procedures which are called at every iteration in each run. For simplicity, the following procedures are described for an active user, $A$ (see figure 5.4).
Check for Neighbourhood (1)

Neighbourhood of $A$

Compute Similarities (2)

If more similar than the current best friend

Update Best Friend (3)
Update Neighbourhood Best (4)
Update Outsiders (5)

Final neighbourhood of $A$
for this iteration

Figure 5.4: The process of neighbourhood selection for the active user, $A$. 

158
Check for Neighbourhood

Each user has to check to see if any users are within close proximity. Close proximity or neighbourhood is defined as any location that lies within a specified range, \( r \). Figure 5.5 below illustrates this process.

![Figure 5.5: The neighbourhood of the active user, \( A \)](image)

The neighbourhood of user \( A \) is defined by the circle with \( A \) at the centre. The radius, \( r \) represents the maximum distance allowed for 2 users to be 'interacting' with each other. More precisely, the active user can only see the users whose position is within the neighbourhood. Thus, the users in \( A \)'s neighbourhood are \( B, C \) and \( E \). Even though \( C \) is in \( A \)'s neighbourhood, he can only interact with \( A \), not \( B \) or \( E \) as they are outside his own neighbourhood (illustrated by a dotted circle). In real life, one 'interacts' with people that one meets. These can be family members, friends, acquaintances or even someone that you have not met before. This process is simply modelling this real world situation. Users belonging in the neighbourhood of the active user are the people that the active user 'meets' and 'interacts' with at this point in time i.e. the current iteration.

As described earlier, the swarming space is unbounded. Special care must be taken when the system is implemented to handle the case where users are at the 'edge' of the space as the 4 corners are actually one single point, see figure 5.3. Figure 5.6 below shows that users \( B \) and \( C \) belong to the neighbourhood of \( A \).
Figure 5.6: Check for Neighbourhood – special case

(2) **Compute Similarities**

The active user has to compute a similarity value, $\text{euclidean}(A, j)$, between himself and each user $j$ that he encounters in his neighbourhood, using a Euclidean function below. Note that only the values of the first 4 features: rating, age, gender and occupation (as these have been found to be efficient) are used in the calculation and the notion of feature weights, $w_f$, is temporary removed from the equation.

$$
\text{euclidean}(A, j) = \frac{1}{z \sum_{i \in A} \sum_{j \in J}} \sum_{i \in A} \sum_{j \in J} \text{diff}_i(A, j)^2 \quad [8]
$$

where:
- $A$ is the active user
- $j$ is a user provided by the profile selection process, where $j \neq A$
- The common items that users $A$ and $j$ have rated are defined as the set $\lambda_1 \ldots \lambda_z$
- $z$ is the number of common movies.
- $i$ is a common movie item, where $\text{profile}(A, i)$ and $\text{profile}(j, i)$ exists.
- $\text{diff}_i(A, j)$ is the difference in profile value for feature $f$ between users $A$ and $j$ on movie item $i$. 


For the first feature, movie rating, the average difference in ratings for all common movies is used. All feature values are normalised so that they lie within the range of 0 and 1. Note that the difference for the gender and occupation features is either 0 (the same) or 1 (different). The similarity measure, \textit{euclidean}(A,j), returns -1 if there are no common movies between two users. Otherwise, the value returned by the Euclidean function lies in the range of 0 to 2, with 0 representing a perfect match to the active user and 2 meaning totally dissimilar where the ratings given by the two users on common items are at opposite extremes i.e. \( x \) gives a rating of 1 and \( y \) a rating of 5 for the same movie.

(3) **Update Neighbourhood Best (nbest)**

Through interaction with someone, one can usually ‘learn’ more about the person. For instance, you meet someone for the first time and perhaps, after having spoken to him/her you decide that you have many things in common and would like to keep in touch.

Applying this idea to the algorithm, the active user \( A \) finds the user in his current neighbourhood that is most similar to him (with the lowest Euclidean value) and uses this user’s current position as the \( nbest \) (neighbourhood best) location. This replaces the \( gbest \) value in the previous chapter. The \( nbest \) location is updated at each iteration and depends on which users belong to the neighbourhood at that point in time. By keeping track of the best user that the active user has just ‘met’, it allows him to keep in touch with this user, i.e. by moving towards his/her location, there is a high probability that he/she will be in the neighbourhood again. Note that the \( nbest \) location lasts for a single iteration and therefore only influences the direction in which the active user moves for the following iteration. After the position of the user is updated, the \( nbest \) location is cleared. If at any point in time, the neighbourhood is empty, the active user just moves towards his best friend (described in the ‘Update Best Friend’ below).

(4) **Update Best Friend (pbest)**

The best friend list is initially empty when the user first joins the system. If the list is empty then the current \( nbest \) user is chosen as the active user’s best friend. However, if both best friend list and neighbourhood are empty, \( pbest \) is set to be the central position of all users in the space. This rule is introduced to allow the active user to move towards the centre of the ‘community’ and have a higher chance of “meeting new people”. If the best friend already exists and he/she is not the same as the \( nbest \) user, then the active user
checks if this \textit{nbest} user is more similar to him than the current best friend. If that is the case then the \textit{nbest} user becomes his new ‘best friend’. However, if the same best friend remains most similar then the \textit{pbest} location is updated with the best friend’s current location.

When there is only one best friend in the list, a force is applied which results in the active user moving towards the best friend, see figure 5.7.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{attraction_to_one_best_friend}
\caption{Attraction to one best friend}
\end{figure}

If, however, there is more than one best friend in the list, the force adjusts the velocity to move towards the central location of all best friends, see figure 5.8.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{attraction_to_multiple_best_friends}
\caption{Attraction to multiple best friends}
\end{figure}

(5) \textbf{Update Outsiders}

For each iteration, the system looks to see if there is an outsider in the list. If the list is not empty, then a \textit{repulsive force} is recalculated. Whereas the relation of best friends is not symmetric, that of outsiders is; if \textit{x} is \textit{y}’s outsider then \textit{y} is also \textit{x}’s outsider. The repulsive force is similar to the concept of ‘charge’ introduced by Blackwell and Bentley (2002a) to deal with collision avoidance. In this work, the repulsive force is used to reduce the number of outsiders appearing in the neighbourhood. This force is another rule added to the PSO dynamic so that the active user would avoid getting close to users that are dissimilar to him.
When there is only one outsider in the list, a repulsive force is applied which results in the active user moving away from the outsider, see figure 5.9.

![Figure 5.9: Repulsive Force with one outsider](image)

If, however, there is more than one outsider in the list, the force adjusts the velocity to move in the direction opposite to the central location of all outsiders. Figure 5.10 illustrates this.

![Figure 5.10: Repulsive Force with multiple users](image)

### 5.1.5. The ClusterPSO Algorithm

The ClusterPSO algorithm is shown below in figure 5.11.
-LOOP 1 for each active user, A, in current Profile Selection set
  Create profile(A)
  Initialise a random initial position for A
  Create empty best friend and outsider list for A
-LOOP 1
  LOOP 2
  LOOP 3 for each active user, A, in current Profile Selection set
  Create an empty neighbourhood for A
  Reset nbest and outsider list for A
  Add users within range of A to neighbourhood
  if A’s neighbourhood is not empty
    LOOP 4 for each user j in A’s neighbourhood
      Compute similarity value between A and j, euclidean(A,j) - training items only
      if similarity value equals -1
        Remove j from neighbourhood
        Add j to A’s outsider list
    END LOOP 4
  Set A’s nbest to be user in A’s neighbourhood that is most similar to A
  Set A’s Xbest in Rule 1 to be nbest’s current position
  if A’s best friend list is empty
    Add nbest to best friend list
    Set A’s Xbest in Rule 1 to be nbest’s current position
  else
    if nbest is more similar than A’s current best friend
      Replace A’s current best friend with nbest
      Set A’s Xbest in Rule 1 to be nbest’s current position
    else
      Update A’s Xbest in Rule 1 to be the best friend’s current position
      if A’s outsider list is not empty
        Set A’s Xavoid in Rule 2 to be the central position of all users in the list
        Set repel to true
      else
        Set repel to false
        if (A’s neighbourhood is empty)
          if A’s best friend list is empty
            Set A’s Xbest in Rule 1 to be the central position of all users in the system
          else
            Update A’s Xbest in Rule 1 to be the best friend’s current position
            Set a flag that A’s Xbest is not used in this iteration
            Set repel to false
          Compute velocity according to Rule 1
          if repel is true
            Update velocity according to Rule 2
            Check velocity limit according to Rule 3
            Update current position of A according to Rule 4
            Reset nbest and outsider list for A
          else
            Increment the number of iterations by 1
            UNTIL maximum iteration is reached (END LOOP 2)
        END LOOP 3
    END LOOP 5 for each active user, A, in current Profile Selection set
    Create an empty neighbourhood for A
    Add users within range of A to neighbourhood
    if A’s neighbourhood is not empty
      LOOP 6 for each user j in A’s neighbourhood
        Compute similarity value between A and j, euclidean(A,j) - training items only
        if similarity value equals -1
          Remove j from A’s final neighbourhood
      END LOOP 6
    if A’s neighbourhood is not empty
      Compute the predicted rating for i using A’s final neighbourhood
      else
        Compute the predicted rating for i using A’s mean rating
      if the difference between predicted and actual ratings for i is 0
        Increment the number of correct predictions (zero tolerance)
      else if the difference between predicted and actual ratings for i is either 1 or 0
        Increment the number of correct predictions (at-most-one tolerance)
    END LOOP 7
  END LOOP 5
END LOOP 5

Figure 5.11: The algorithm describing the ClusterPSO recommender system

From figure 5.11 above, the algorithm starts by creating profiles of all active users, A. This task, see loop 1 from line 1 to line 5, is referred to as the profile generator, see section 3.2.1. Users that participate in each run are selected using a method of profile
selection, described in section 3.2.2, to form the current profile selection set (lines 1, 7 and 50). Loop 2, starting at line 6 and ending at line 49, represents the learning process where the ClusterPSO algorithm is responsible for the movement of all active users in the system. Within this loop, the neighbourhood selection process, described earlier in 5.1.4, is employed to select the neighbourhood and update the nbest, pbest and outsiders lists of the active users at each iteration (lines 8 to 40). Note that the learning process is carried out for all active users within a single loop (loop 2). This differs from the previous GA and PSO systems whose learning process has to be repeated for each active user separately. Loop 5 of figure 5.11 is responsible for selecting the final neighbourhood of users similar to each \( A \) (refer to lines 51 to 58) and predicting the ratings for the movie items in the test set of \( A \) (loop 7 starting at line 59 and ending at line 68). The accuracy of the predictions is then recorded to evaluate the performance of the system, see Experiments section.

The swarming dynamics are similar to those used in the PSO Recommender system. However, for an iteration where the list of outsiders is not empty, another rule is added to handle the repulsive force, see Rule 2. Thus, the new dynamics become:

\[
\begin{align*}
\dot{v}_i &= w\dot{v}_i + c_1 p_i (x_{pbest,i} - x_i) + c_2 p_i (x_{nbest,i} - x_i) \\
\dot{v}_i &= v_i - f_3 r_2 (x_{void,i} - x_i) \\
&\text{if } (|v_i| > v_{\text{max}}) \quad v_i = (v_{\text{max}} / |v_i|) v_i \\
&\text{Rule 1} \\
&\text{Rule 2} \\
&\text{Rule 3} \\
&\text{Rule 4}
\end{align*}
\]

where:

\( x_i \) is the current position of user \( i \)

\( x_{pbest,i} \) is the centre position of current positions of all best friends of user \( i \)

\( x_{nbest,i} \) is the centre position of current positions of the best users in the neighbourhood of user \( i \)

\( v_i \) is the velocity of user \( i \)

\( w \) is a random inertia weight between 0.5 and 1 (Eberhart and Shi 2001b)

\( c_1, c_2 \) and \( c_3 \) are spring constants whose values are set to 1.494 (Eberhart and Shi 2001b)

\( r_1, r_2 \) and \( r_3 \) are random numbers between 0 and 1 (Blackwell and Bentley 2002a)

\( f_3 \) is the repulsive factor

\( x_{void,i} \) represents the central position of all outsiders in user \( i \)'s neighbourhood. A repulsive factor, denoted \( f_3 \) measures the impact that the repulsive force has on velocity; the higher the value the greater that dissimilar users would repel each other.
5.1.6. Making Recommendation

The recommendation process is again similar to the one described in section 3.2.3. The final neighbourhood of friends for each active user also includes the user's best friends and excludes any outsiders which may be present. Again, the predicted rating equation used by the GA and PSO systems is also employed here.

\[
predicted\_rating(A,i) = mean_A + k \sum_{j=1}^{n} \text{euclidean}(A, j)(\text{rating}(j, i) - mean_j) \ [1, \ p.33]
\]

where:
- \(mean_j\) is the mean rating for user \(j\)
- \(k\) is a normalising factor such that the sum of the Euclidean distances is equal to 1.
- \(\text{rating}(j, i)\) is the actual rating that user \(j\) has given on item \(i\), if \(\text{rating}(j, i)\) exists.
- \(n\) is the size of the neighbourhood.

As with the GA and PSO recommenders, users that are most similar to the active user should provide a greater contribution than less similar ones in the predicted rating equation. Therefore, the calculated Euclidean distances have to be reversed i.e. 0 maps to 2 and 2 maps to 0. The function used to do this is simply, \(f(x) = 2 - x\).

5.2. Preliminary Cluster Analysis

Prior to running the ClusterPSO experiments, the following investigations were carried out in order to gain a preliminary understanding of how users should be grouped. This allows us to evaluate the effectiveness of a clustering algorithm.

5.2.1. Manual Allocation

It was first decided to group similar users manually. This would be used to compare against the final neighbourhoods obtained by ClusterPSO.

Table 5.1 below shows the similarity values for the first 10 active users computed using the Euclidean function described earlier in Section 5.1.4 (recall that this does not use the notion of feature weights). The closer the value to zero, the more similar the users are.
As mentioned earlier, a value of -1 is reserved to denote that a pair of users is considered to be dissimilar to each other as they have no common movies.

<table>
<thead>
<tr>
<th>User</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>N/A</td>
<td>1.52</td>
<td>1.03</td>
<td>0.50</td>
<td>1.45</td>
<td>1.06</td>
<td>1.20</td>
<td>1.03</td>
<td>-1.00</td>
<td>1.10</td>
</tr>
<tr>
<td>2</td>
<td>1.52</td>
<td>N/A</td>
<td>1.51</td>
<td>1.79</td>
<td>0.30</td>
<td>1.44</td>
<td>1.44</td>
<td>-1.00</td>
<td>1.46</td>
<td>1.41</td>
</tr>
<tr>
<td>3</td>
<td>1.03</td>
<td>1.51</td>
<td>N/A</td>
<td>1.03</td>
<td>-1.00</td>
<td>1.04</td>
<td>-1.00</td>
<td>1.14</td>
<td>-1.00</td>
<td>1.13</td>
</tr>
<tr>
<td>4</td>
<td>0.50</td>
<td>1.79</td>
<td>1.03</td>
<td>N/A</td>
<td>1.42</td>
<td>-1.00</td>
<td>-1.00</td>
<td>-1.00</td>
<td>-1.00</td>
<td>-1.00</td>
</tr>
<tr>
<td>5</td>
<td>1.45</td>
<td>0.30</td>
<td>-1.00</td>
<td>1.42</td>
<td>N/A</td>
<td>1.42</td>
<td>1.54</td>
<td>1.44</td>
<td>-1.00</td>
<td>1.49</td>
</tr>
<tr>
<td>6</td>
<td>1.06</td>
<td>1.44</td>
<td>1.04</td>
<td>-1.00</td>
<td>1.42</td>
<td>N/A</td>
<td>1.07</td>
<td>1.12</td>
<td>1.09</td>
<td>1.04</td>
</tr>
<tr>
<td>7</td>
<td>1.20</td>
<td>1.44</td>
<td>-1.00</td>
<td>-1.00</td>
<td>1.54</td>
<td>N/A</td>
<td>1.07</td>
<td>0.48</td>
<td>1.16</td>
<td>1.02</td>
</tr>
<tr>
<td>8</td>
<td>1.03</td>
<td>-1.00</td>
<td>1.14</td>
<td>-1.00</td>
<td>1.44</td>
<td>N/A</td>
<td>1.12</td>
<td>0.48</td>
<td>1.16</td>
<td>1.01</td>
</tr>
<tr>
<td>9</td>
<td>-1.00</td>
<td>1.46</td>
<td>-1.00</td>
<td>-1.00</td>
<td>-1.00</td>
<td>1.09</td>
<td>1.16</td>
<td>1.01</td>
<td>N/A</td>
<td>-1.00</td>
</tr>
<tr>
<td>10</td>
<td>1.10</td>
<td>1.41</td>
<td>1.13</td>
<td>-1.00</td>
<td>1.49</td>
<td>1.04</td>
<td>1.02</td>
<td>1.12</td>
<td>-1.00</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Table 5.1: Similarity table

Table 5.1 shows how similar/dissimilar users are and thus, whether they should appear in the same neighbourhood. By inspection, table 5.2 was derived.

<table>
<thead>
<tr>
<th>User</th>
<th>Similar (best friends)</th>
<th>Dissimilar (outsiders)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>1,4</td>
<td>5,7,9</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>6,7,8,9,10</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>3,9</td>
</tr>
<tr>
<td>6</td>
<td>10</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>3,4</td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>2,4</td>
</tr>
<tr>
<td>9</td>
<td>8</td>
<td>1,3,4,5,10</td>
</tr>
<tr>
<td>10</td>
<td>7</td>
<td>4,9</td>
</tr>
</tbody>
</table>

Table 5.2: User preference table for the first 10 users

Figure 5.12 provides a visual representation of users and their best friends using the information from the above table. An arrow from user \( X \) to user \( Y \) means that \( Y \) is considered most similar to \( X \). Therefore, it is expected that user \( X \) would move towards user \( Y \) so that \( Y \) can belong in \( X \)'s neighbourhood.
Extending the above graph to include the outsider information, where dissimilar users distance themselves from each other, we can generate a graph showing the various cluster groups and their relation to one another. For example, user 9 should be close to user 8 whilst avoiding users 1, 3, 4, 5 and 10. Figure 5.13 shows a possible set of neighbourhoods: \{N1: 1,3,4\} \{N2: 2,5\} \{N3: 6,7,8,10\} \{N4: 9\}.

The same process was applied to the first 50 users. Table 5.3 below shows the user preference and figure 5.14 provides a visual representation of users and their best friends.
Therefore, manual allocation is only possible when the number of users is small. As the number of users increases, this method becomes increasingly impractical. For this reason, further investigation was carried out using an existing constrained clustering algorithm.

<table>
<thead>
<tr>
<th>User</th>
<th>Similar</th>
<th>Distimilar</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>44</td>
<td>169</td>
</tr>
<tr>
<td>2</td>
<td>11,12</td>
<td>20,22,23</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>20,22,23</td>
</tr>
<tr>
<td>4</td>
<td>11,12</td>
<td>20,22,23</td>
</tr>
<tr>
<td>5</td>
<td>11,12</td>
<td>20,22,23</td>
</tr>
<tr>
<td>6</td>
<td>11,12</td>
<td>20,22,23</td>
</tr>
<tr>
<td>7</td>
<td>11,12</td>
<td>20,22,23</td>
</tr>
<tr>
<td>8</td>
<td>11,12</td>
<td>20,22,23</td>
</tr>
<tr>
<td>9</td>
<td>11,12</td>
<td>20,22,23</td>
</tr>
<tr>
<td>10</td>
<td>11,12</td>
<td>20,22,23</td>
</tr>
<tr>
<td>11</td>
<td>11,12</td>
<td>20,22,23</td>
</tr>
<tr>
<td>12</td>
<td>11,12</td>
<td>20,22,23</td>
</tr>
<tr>
<td>13</td>
<td>11,12</td>
<td>20,22,23</td>
</tr>
<tr>
<td>14</td>
<td>11,12</td>
<td>20,22,23</td>
</tr>
<tr>
<td>15</td>
<td>11,12</td>
<td>20,22,23</td>
</tr>
<tr>
<td>16</td>
<td>11,12</td>
<td>20,22,23</td>
</tr>
<tr>
<td>17</td>
<td>11,12</td>
<td>20,22,23</td>
</tr>
<tr>
<td>18</td>
<td>11,12</td>
<td>20,22,23</td>
</tr>
<tr>
<td>19</td>
<td>11,12</td>
<td>20,22,23</td>
</tr>
<tr>
<td>20</td>
<td>11,12</td>
<td>20,22,23</td>
</tr>
<tr>
<td>21</td>
<td>11,12</td>
<td>20,22,23</td>
</tr>
<tr>
<td>22</td>
<td>11,12</td>
<td>20,22,23</td>
</tr>
<tr>
<td>23</td>
<td>11,12</td>
<td>20,22,23</td>
</tr>
<tr>
<td>24</td>
<td>11,12</td>
<td>20,22,23</td>
</tr>
<tr>
<td>25</td>
<td>11,12</td>
<td>20,22,23</td>
</tr>
<tr>
<td>26</td>
<td>11,12</td>
<td>20,22,23</td>
</tr>
<tr>
<td>27</td>
<td>11,12</td>
<td>20,22,23</td>
</tr>
<tr>
<td>28</td>
<td>11,12</td>
<td>20,22,23</td>
</tr>
<tr>
<td>29</td>
<td>11,12</td>
<td>20,22,23</td>
</tr>
<tr>
<td>30</td>
<td>11,12</td>
<td>20,22,23</td>
</tr>
<tr>
<td>31</td>
<td>11,12</td>
<td>20,22,23</td>
</tr>
<tr>
<td>32</td>
<td>11,12</td>
<td>20,22,23</td>
</tr>
<tr>
<td>33</td>
<td>11,12</td>
<td>20,22,23</td>
</tr>
<tr>
<td>34</td>
<td>11,12</td>
<td>20,22,23</td>
</tr>
<tr>
<td>35</td>
<td>11,12</td>
<td>20,22,23</td>
</tr>
<tr>
<td>36</td>
<td>11,12</td>
<td>20,22,23</td>
</tr>
<tr>
<td>37</td>
<td>11,12</td>
<td>20,22,23</td>
</tr>
<tr>
<td>38</td>
<td>11,12</td>
<td>20,22,23</td>
</tr>
<tr>
<td>39</td>
<td>11,12</td>
<td>20,22,23</td>
</tr>
<tr>
<td>40</td>
<td>11,12</td>
<td>20,22,23</td>
</tr>
<tr>
<td>41</td>
<td>11,12</td>
<td>20,22,23</td>
</tr>
<tr>
<td>42</td>
<td>11,12</td>
<td>20,22,23</td>
</tr>
<tr>
<td>43</td>
<td>11,12</td>
<td>20,22,23</td>
</tr>
<tr>
<td>44</td>
<td>11,12</td>
<td>20,22,23</td>
</tr>
<tr>
<td>45</td>
<td>11,12</td>
<td>20,22,23</td>
</tr>
<tr>
<td>46</td>
<td>11,12</td>
<td>20,22,23</td>
</tr>
<tr>
<td>47</td>
<td>11,12</td>
<td>20,22,23</td>
</tr>
<tr>
<td>48</td>
<td>11,12</td>
<td>20,22,23</td>
</tr>
<tr>
<td>49</td>
<td>11,12</td>
<td>20,22,23</td>
</tr>
<tr>
<td>50</td>
<td>11,12</td>
<td>20,22,23</td>
</tr>
</tbody>
</table>

Table 5.3: User preference table for the first 50 users

Unfortunately, as should be evident from the complexity of figure 5.14 below, it was not feasible to work out possible neighbourhoods manually when the outsiders information was incorporated.

Therefore, manual allocation is only possible when the number of users is small. As the number of users increases, this method becomes increasingly impractical. For this reason, further investigation was carried out using an existing constrained clustering algorithm.
5.2.2. Conventional Clustering

Most conventional clustering algorithms cannot be used to solve this particular problem as they have no notion of an exclude list.

There is however a clustering algorithm, COP_KMEANS (Wagstaff et al. 2001), which is based on the $k$-means algorithm but which also takes into account constraints. These constraints usually describe which data items should or should not be grouped together. In this thesis, only the latter constraints which prevent dissimilar users from being grouped together are applied. COP_KMEANS algorithm is shown below in figure 5.15.
Randomly select \(C_1, \ldots, C_k\) to be initial cluster centres.

Loop

- Loop for each data item \(d_i\) in the data set
  - Assign \(d_i\) to the closest cluster centre \(C_j\) of cluster \(j\) where \(\text{violate\_constraints}(d_i, j)\) is false. If no such cluster exists then fail (and return empty partition).

End loop

- Loop for each cluster \(j\)
  - Update cluster centre \(C_j\) by averaging all \(d_i\) in \(j\).

End loop

Until convergence (the maximum number of iterations is reached or the same data items are assigned to each cluster for a number of iterations)

Figure 5.15: COP-KMEANS algorithm

\(\text{violate\_constraints}(d_i, j)\) returns \text{true} if there exists another data item \(d_a\) in cluster \(j\) which cannot be in the same cluster as \(d_i\) but \(d_a\) is already assigned to \(j\).

A recommender system based on a modified COP_KMEANS was implemented. Each user was represented by a vector with \(n\) components where \(n\) is the number of items in the system and the \(i^{th}\) component corresponds to the user's rating for item \(i\). The distance between each user was computed using the standard Euclidean function. By running this algorithm, \(k\) non-overlapping clusters containing similar users were produced where \(k\) is the specified number of clusters.

Experiments with COP_KMEANS

Experiments 1 and 2 were carried out by running COP_KMEANS for the first 10 and 50 users, respectively, to obtain \(k\) number of clusters. Note that initial experiments were performed with the value of \(k\) varying from 1 to 5 for experiment 1 and from 1 to 15 for experiment 2. This was to determine the most appropriate values of \(k\) (that return resulting clusters most similar to those expected, see tables 5.2 and 5.3) to be used in the experiments. As a result, \(k\) was initially set to 4 for experiment 1 and 15 for experiment 2. Additionally, at the start of each run, \(k\) users were randomly selected to be cluster centres. Once all users were assigned, these groupings were used to compute the predicted rating for each active user.

The best results from 30 runs, see figures 5.16 to 5.19, were obtained for both zero and at-most-one tolerance and compared against those obtained from the Flocking PSO presented in section 4.2.3.
Figure 5.16: Comparison between COP_KMEANS and Flocking PSO - best results for experiment 1 with zero tolerance.

Figure 5.17: Comparison between COP_KMEANS and Flocking PSO - best results for experiment 1 with at-most-one tolerance.

Figure 5.18: Comparison between COP_KMEANS and Flocking PSO - best results for experiment 2 with zero tolerance.
Analysis of COP_KMEANS

It can be seen that results for both COP-KMEANS and Flocking PSO were similar for experiment 1, see figures 5.16 and 5.17. Looking at the resulting clusters of users for the first 10 users, it was seen that for all 30 runs, COP_KMEANS was able to assign all 10 users into one of the clusters without violating the constraints i.e. dissimilar users in the same cluster. Resulting clusters obtained from runs 3 and 9 are as follows:

**Run 3**  
{Cluster 1: 7,9} {Cluster 2: 3,4} {Cluster 3: 1,8,6} {Cluster 4: 2,5,10}

**Run 9**  
{Cluster 1: 7,8} {Cluster 2: 2,9} {Cluster 3: 1,3,6,10} {Cluster 4: 4,5}

For run 3 above, the system assigned user 3 with his best friend, 4, and users 2 and 5 with each other. The cluster allocation was perhaps better for run 9 as users 7 and 8 were assigned to be together and similarly, user 3 belonged to the same cluster as his best friend, 1, and user 6 was with 10. However, as a user can only appear in a single cluster, it was not possible for all active users to be assigned to the same cluster as their best friends, see table 5.2.

Looking at the results for experiment 2 with 50 users, it can be seen that when Flocking PSO performed better, the difference in prediction accuracy between the two systems in some cases was large (as great as 43%), see figures 5.18 and 5.19. Nonetheless, COP_KMEANS was able to obtain higher prediction accuracy than Flocking PSO for a number of active users. As it was shown in chapter 4 that the PSO recommender system...
outperformed the Pearson algorithm for most users, this demonstrates that a constrained clustering technique is also better at selecting users to give recommendations than the Pearson algorithm.

Looking at the resulting clusters obtained for experiment 2 with the number of clusters, $k$, set to 15, it was found that most users were able to be assigned to a cluster. However, it was apparent that at each run there were still a few users that could not be assigned to any of the clusters. An example of the resulting clusters obtained from run 28 is shown below.

**Run 28**

- Cluster 1: 49, 17, 6
- Cluster 2: 14, 8, 44
- Cluster 3: 45, 28, 20
- Cluster 4: 46, 33, 35, 19
- Cluster 5: 38, 4, 5
- Cluster 6: 15, 50, 10
- Cluster 7: 16, 41, 11
- Cluster 8: 9, 25, 18, 43
- Cluster 9: 37, 7, 1
- Cluster 10: 39, 36
- Cluster 11: 22, 2, 12
- Cluster 12: 13, 47, 31
- Cluster 13: 29, 21, 40, 24
- Cluster 14: 30, 23, 48
- Cluster 15: 32, 3, 26
- (Unassigned: 34, 27)

It became clear that it was almost impossible to assign users with their best friends whilst avoiding outsiders due to the increase in the number of constraints, see table 5.3. Because of the increase in the number of clusters, the average number of users per cluster remained similar to that in experiment 1, thus the prediction accuracy for experiment 2 did not improve for most users. For this reason, the number of clusters, $k$, was reduced from 15 to 5 in an attempt to increase the number of users per cluster. The following clusters were then obtained:

- Cluster 1: 4, 19, 5, 44
- Cluster 2: 40, 24, 17
- Cluster 3: 13, 30, 2, 3, 26, 6
- Cluster 4: 7, 38, 22, 8, 23, 1, 16, 42, 18, 10, 43, 11, 20, 12
- Cluster 5: 21, 14, 15, 9, 32, 49, 25, 28
- (Unassigned: 56, 37, 29, 46, 47, 39, 48, 31, 33, 41, 50, 34, 27, 35, 36)

By increasing the number of users per cluster, this allowed 4 users (users 3, 4, 21 and 32) to have their best friend assigned to the same cluster. Moreover, the prediction accuracy was found to improve for a number of active users. However, this also resulted in the system being unable to assign 15 users to any existing cluster.

This accentuates the complexity of the neighbourhood selection process in recommender systems. Furthermore, COP_KMEANS is not adaptive; any changes made by users would require a cluster reallocation which involves re-running the system for all users in
the system. This can incur a huge computational cost if the number of users is large and hence the problem of user scalability can arise.

For 10 active users, a typical run by the COP_KMEANS system took approximately 6 seconds (if stabilised before the maximum number of iterations was reached) and 42 seconds (if the maximum number of iterations was reached). Similarly, for 50 active users, a typical run took approximately 370 seconds (if stabilised) and 690 seconds (if the maximum number of iterations was reached). A comparison of computational speed is shown later in section 6.7.

5.3. Experiments

Experiments were conducted with various parameter values (see individual experiments). As mentioned above, the users in the ClusterPSO system move simultaneously to find their own neighbourhood of friends. Experiments in this chapter were therefore conducted in a slightly different manner than from the previous two chapters.

The users employed for each experiment were still the same as before, they are:

**Experiment 1:** The first 10 users (from 944) acted as active users, $A$ (see lines 1, 7 and 50 of figure 5.11) and also formed the Profile Selection set to provide recommendations for one another (see lines 10 to 17 and lines 52 to 58 of figure 5.11).

**Experiment 2:** The first 50 users (from 944) acted as active users, $A$ and also formed the Profile Selection set to provide recommendations for one another. Note that the 10 users in experiment 1 were a subset of the 50 users here.

For experiments 1 and 2, by the end of a run, each active user would have determined his final neighbourhood of friends. The system then used these friends to predict the rating of all test movies for the active user. Because the neighbourhood selection was performed simultaneously, the system was run 30 times (for both 10 and 50 users) in order to obtain the results of 30 runs.

**Experiment 3:** Each of the first 10 users was selected as the active user, $A$, in turn. For each run, 9 different users were selected randomly (out of 944) to join the current active
user and form the Profile Selection set (to provide recommendations). All three systems used the same active user and the 9 randomly selected users at each run.

**Experiment 4:** Each of the first 50 users was selected as the active user, A, in turn. For each run, 49 different users were selected randomly (out of 944) to join the current active user and form the Profile Selection set (to provide recommendations). All three systems used the same active user and the 49 randomly selected users at each run. Note that the 9 randomly selected users from experiment 3 were a subset of the 49 users here.

Experiments 3 and 4 took significantly longer to complete than the first two experiments. Each of the first 10 or 50 users had to be picked as the active user in turn. After each run, only the result for the current active user was recorded and the rest (random users) were discarded. Therefore, to obtain the results of 30 runs, the system had to be run 300 times for 10 users (30 runs for each user) and 1500 times for 50 users.

Four system variables were evaluated to assess their effect on system performance: swarming space size, maximum range of neighbourhood, maximum velocity and repulsive factor.

Table 5.4 below shows the initial experimental parameter values. These values were kept the same in all experiments (unless otherwise specified):

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of users</td>
<td>10 or 50</td>
</tr>
<tr>
<td>(The number of users in the swarming space at each iteration)</td>
<td></td>
</tr>
<tr>
<td>maximum number of iterations for each run</td>
<td>1000</td>
</tr>
<tr>
<td>(The iteration at which the current neighbourhood for each active user is used to give recommendations)</td>
<td></td>
</tr>
<tr>
<td>swarming space size</td>
<td>1500</td>
</tr>
<tr>
<td>(The width and length of the swarming space)</td>
<td></td>
</tr>
<tr>
<td>maximum range of neighbourhood</td>
<td>100</td>
</tr>
<tr>
<td>(The radius around the active user which is considered to be the user’s neighbourhood)</td>
<td></td>
</tr>
<tr>
<td>maximum velocity</td>
<td>50</td>
</tr>
<tr>
<td>(The maximum velocity allowed at each iteration)</td>
<td></td>
</tr>
<tr>
<td>repulsive factor (f3 in rule 2)</td>
<td>2</td>
</tr>
</tbody>
</table>
The purpose of the experiments was to obtain a set of values most suitable for each of the four system variables that would maximise the performance of the ClusterPSO recommender system in terms of prediction accuracy. Note that full experimental results and analysis are shown in Appendix B.

From the experiments, the following results were found:

- **The swarming space size**: it was found that the size of 1500 units worked well for 10 users in experiments 1 and 3. However, when the number of users was increased to 50, a problem of overcrowding was observed. To overcome this, the swarming space size was increased and the size of 7500 units was found to be most suitable for 50 users in experiments 2 and 4. It is therefore paramount that the correct swarming space size is chosen, too small and the space is overcrowded, too large and the users are too widely scattered which would leave many users being unable to find their neighbourhood to provide recommendations.

- **The maximum range of neighbourhood**: A constraint applied to this variable is that the maximum range cannot exceed the swarming space size. Moreover, if the variable is set to be the same value as the width or length of the space, it means that everyone can see each other and that all users would belong to the same neighbourhood. It is therefore important to set the maximum range to the right proportion of the swarming space in order for good neighbourhoods to be formed. From the results, it was found that the maximum range of 150 units generally worked well in all four experiments.

- **The maximum velocity**: in terms of prediction accuracy, it was found that the results obtained from varying the maximum velocity values were similar. Nonetheless, the value of 50 units was shown to perform the best for all four experiments. However, further analysis using the 3D visualisation tool revealed that the value of 200 units was actually better at clustering users according to the user preferences tables in the Preliminary Cluster Analysis section. So far, the assumption has been that if a system is able to find a neighbourhood of users most similar to each active user (i.e. according to user preferences tables), the
prediction accuracy obtained should be significantly high. As the results from setting the maximum velocity value to 200 units did not outperform those when set to 50 units, this therefore suggests that the current similarity measure without the notion of feature weights needs to be revised.

- **The repulsive factor:** the results obtained from varying the value of the repulsive factor did not show a great difference. In terms of prediction accuracy, the repulsive factor of 1 was seen to perform well for all experiments. Again, looking at the visualisation of user dynamics, it was found that users were grouped according to the user preferences tables when the repulsive factor was set to 2. Because the results obtained from the repulsive factor of 2 did not outperform those obtained with other factor values, this further suggests the need to re-examine the similarity measure and that the notion of feature weights should be reintroduced.

5.3.1. Comparisons with Previous Systems: Pearson algorithm, Genetic algorithm, Particle Swarm Optimisation and COP_KMEANS

Figures 5.20 to 5.27 below show the comparisons of the results in experiments 1 and 2 obtained from each of the previous systems against those from ClusterPSO. Note that only the first two experiments were considered as these were the ones where all active users ran simultaneously in ClusterPSO. In this section, only the best results are examined and further analysis using average and worst results will be carried out in chapter 6.

![Figure 5.20: Comparison between PA and ClusterPSO - best results for both algorithms for experiment 1 with zero tolerance](image)

Figure 5.20: Comparison between PA and ClusterPSO - best results for both algorithms for experiment 1 with zero tolerance
Figure 5.21: Comparison between PA and ClusterPSO - best results for both algorithms for experiment 2 with zero tolerance

Figure 5.22: Comparison between GA and ClusterPSO - best results for both algorithms for experiment 1 with zero tolerance

Figure 5.23: Comparison between GA and ClusterPSO - best results for both algorithms for experiment 2 with zero tolerance
Figure 5.24: Comparison between Flocking PSO and ClusterPSO - best results for both algorithms for experiment 1 with zero tolerance

Figure 5.25: Comparison between Flocking PSO and ClusterPSO - best results for both algorithms for experiment 2 with zero tolerance
Figures 5.20 and 5.21 above show the results obtained from the non-adaptive PA system compared against those from the ClusterPSO recommender. As illustrated, the ClusterPSO outperformed the PA on all but 1 active user for experiment 1 and all but 6 active users for experiment 2.

From figures 5.22 and 5.23, it can be seen that the difference in prediction accuracy between the ClusterPSO and GA systems was smaller than that found against the PA system. The ClusterPSO performed better than the GA for 6 out of 10 active users in experiment 1 and for 29 out of 50 active users in experiment 2.

Overall, the performance of both ClusterPSO and Flocking PSO was similar for experiment 1, see figure 5.24. However, in experiment 2, the prediction accuracy for ClusterPSO increased and was higher than that of the Flocking PSO for 32 out of 50 active users, see figure 5.25. This was expected as it was found in chapter 4 that in the PSO recommender system, when more users were being considered, this sometimes resulted in many less similar users being added to the neighbourhood, and hence, lowered the overall prediction accuracy. This comparison therefore suggests that the ClusterPSO does not experience this problem.

The performance between the ClusterPSO and COP_KMEANS was approximately the same for experiment 1, see figure 5.26. Again, this was expected as both are based on the same principle of clustering those that are similar and avoiding those that are not. However, as the number of users was increased in experiment 2, the number of
constraints (those users that cannot be grouped together) also increased, and as a result, COP_KMEANS failed to assign a number of active users to an existing cluster, see section 5.2.2. Because ClusterPSO did not encounter this problem, as expected, the ClusterPSO outperformed the COP_KMEANS system on all 50 active users in experiment 2, see figure 5.27.

5.4. Summary

This chapter investigated the neighbourhood selection process and presented a novel algorithm, ClusterPSO, that allows the users to dynamically select their own neighbourhood of ‘friends’ and avoid those that are dissimilar to them.

In order to gain a preliminary understanding of how users should be grouped, a manual allocation of users was performed. This manual process is only feasible when the number of users is small. For this reason, a constrained clustering algorithm, COP_KMEANS, was used and the resulting clusters were compared against those obtained manually. When the number of users was small, the algorithm was able to group similar users whilst avoiding dissimilar ones. However, as the number of users was increased, the system was unable to completely assign all users to clusters due to the non-overlapping property of clusters (users can only be assigned to a single cluster). This shows the complexity of the neighbourhood selection process. Moreover, the quality of the resulting clusters depends on the number of clusters and also, the initial cluster centres which are randomly selected. Although, COP_KMEANS was shown to perform well when appropriate parameters were chosen, it is not adaptive. Changes to the user preferences require the clusters to be recomputed, this takes considerable time and is thus, not suitable for an online recommender system.

Numerous experiments were carried out and it was shown that users in ClusterPSO were able to successfully select their own neighbourhood that contained only their most similar users. With the aid of a repulsive force, dissimilar users were automatically excluded. As neighbourhoods for ClusterPSO can overlap, this does not impose a limit to the number of users in the system.

Four system variables in ClusterPSO were investigated to assess their effect on the performance of the system and suitable values were found. However, it was found that
the similarity function currently used does not give an accurate evaluation of the similarity between 2 users. This suggests that the similarity measure needs to be revised and that perhaps, the similarity measure employed with feature weights in chapters 3 and 4 should be reconsidered.

Finally, a comparison between the ClusterPSO and other systems: PA, GA, PSO and COP_KMEANS was presented which showed that the overall performance of the ClusterPSO was as good as or better than that of the other systems.
CHAPTER 6

ClusterWeight: A ClusterPSO algorithm with Feature Weights

The previous chapter described a novel algorithm called ClusterPSO which allows users to dynamically group themselves together based on their similarities. This way the profile matching task can be performed for all users simultaneously. The notion of outsiders was introduced for the first time. These are users that appear in the neighbourhood but do not belong there. Outsiders are considered dissimilar to the active user at this point in time as they do not share any common movies with the active user. When the active user encounters an outsider, he adds this user to the outsider list. The list is used to help the active user avoid these outsiders by moving in the opposite direction, and hence reduce the number of dissimilar users in the neighbourhood.

The experimental results showed that the system worked well, achieving relatively high performance compared to previous systems. However, it was discovered that the similarity function used i.e. a standard Euclidean distance function without the feature weights did not give an accurate evaluation of similarity between 2 users. The system in this chapter, based on the ClusterPSO, brings back the concept of using the feature weights to represent the user recommendation preferences which were used successfully in the GA and PSO systems presented in chapters 3 and 4.

The chapter is organised as follows: section 6.1 describes a modified algorithm of the ClusterPSO recommender system presented in chapter 5. Section 6.2 presents the final algorithm, ClusterWeight, which builds on the modifications to the ClusterPSO, and section 6.3 provides experimental results and analysis. Sections 6.4, 6.5, 6.6 and 6.7 compare the ClusterWeight against the PA, GA, PSO and COP_KMEANS recommender systems respectively. Section 6.8 describes a pilot study carried out with real participants. Finally, section 6.9 concludes.

6.1. Modified ClusterPSO

It is thought that similar users are likely to have similar recommendation preferences i.e. similar sets of weights. For this reason, the earlier concept of similar users being
attracting to each other is still used in the modified ClusterPSO but the similarity measure now takes into account these weights in order to give a more accurate evaluation of similarity between two users. The ClusterPSO algorithm in chapter 5 is shown below in figure 6.1.

Figure 6.1: Modified ClusterPSO algorithm
\[ v_i = w_i + c_1 r_1 (x_{\text{pbest}, i} - x_i) + c_2 r_2 (x_{\text{nbest}, i} - x_i) + c_4 r_4 (x_{\text{bestpos}, i} - x_i) \]  

**Rule 1**

\[ v_i = v_i - f_3 \sqrt{3} r_3 (x_{\text{avoid}, i} - x_i) \]  

**Rule 2**

\[ \text{if} \{ v_i > v_{\text{max}} \} \quad v_i = (v_{\text{max}} / |v_i|) v_i \]  

**Rule 3**

\[ x_i = x_i + v_i \]  

**Rule 4**

where:

- \( x_{\text{bestpos}} \) is the best position that returns the highest similarity value
- \( x_i \) is the current position of user \( i \)
- \( x_{\text{pbest}, i} \) is the centre position of current positions of all best friends of user \( i \)
- \( x_{\text{nbest}, i} \) is the centre position of current positions of the best users in the neighbourhood of user \( i \)
- \( v_i \) is the velocity of user \( i \)
- \( w \) is a random inertia weight between 0.5 and 1
- \( c_1, c_2 \) and \( c_4 \) are spring constants whose values are set to 1.494
- \( r_1, r_2 \) and \( r_4 \) are random numbers between 0 and 1

For this modified system with feature weights, the two major changes to the original ClusterPSO are:

- The swarming space – which is now a multi-dimensional space where each position can be mapped into a set of feature weights, see lines 12 of figure 6.1.
- *Best location* – which is an additional attractor in Rule 1 of the swarming dynamics (see line 4 of figure 6.1).

These modifications are described in more detail below. Note that these changes only occur in the learning stage i.e. loop 2 of figure 6.1 and that the weights are used to determine best location only and not for the recommendation (loop 5 of figure 6.1 remains the same as that of figure 5.11).

### 6.1.1. Swarming Space

The space in this modified system is now a multi-dimensional hyperspace where each axis represents a feature of the dataset. Each position or location in the space therefore represents a different combination of feature weights. The method used to transform a position into a set of weights is the same as that used in chapter 4 (see equation 5 on page 106). The sum of all the weights adds up to one. As it was found that 4 features: rating, age, gender and occupation were most efficient for the PSO Recommender in chapter 4, this system also uses this number of features and hence, the swarming space has 4
dimensions. Each axis still wraps around making the space edge-free, see figures 5.2 and 5.3.

The method for computing the distance between two users to determine whether they are within range of each other has also been modified. This is to take into account the change in the swarming space from 2 to 4 dimensions. For this task, the standard Euclidean distance function to compute a distance between two points in the space is used.

### 6.1.2. Best Location

As mentioned above, best location is an additional attractor in the learning stage which has been added such that the user would remember the location that returns the highest similarity value, see line 30 of figure 6.1. Thus, the equation at line 15 to calculate a similarity value, $\text{euclidean}(A,j)$, between the user himself and each user $j$, is changed back to the one successfully used in chapters 3 and 4 to take into account the 4 feature weights. This is shown below.

$$
\text{euclidean}(A,j) = \frac{1}{z} \sum_{i=1}^{\lambda} \sqrt{\sum_{j=1}^{4} w_f \cdot \text{diff}_f(A,j)}^2
$$

[4, p.84]

where:

- $A$ is the active user
- $j$ is a user provided by the profile selection process, where $j \neq A$
- The common items that users $A$ and $j$ have rated are defined as the set $\lambda_1 \ldots \lambda_z$.
- $z$ is the number of common movies.
- $w_f$ is the active user's weight for feature $f$
- $i$ is a common movie item, where $\text{profile}(A,i)$ and $\text{profile}(j,i)$ exist.
- $\text{diff}_f(A,j)$ is the difference in profile value for feature $f$ between users $A$ and $j$ on movie item $i$.

Note that instead of using -1 to indicate outsiders, the system introduces an outsider threshold so that if the similarity value between the active user and a user $j$ is less than this threshold value, $j$ is considered to be an outsider to the active user.

To illustrate the effect of best location, the following walkthrough example is shown. For this example, assume B is an active user A's best friend (pbest) and neighbourhood best (nbest) and there are no outsiders. For simplicity, the space is 2 dimensional which
represents age and gender. In figure 6.2, if the current position of \( A \) is \((1,1)\), this is mapped to feature weights of \((0.5, 0.5)\) which is then used to calculate similarity value between \( A \) and \( B \). If \( A \)'s age is 0.3 (normalised) and gender is 1 and \( B \)'s age is 0.4 (normalised) and gender is 1, the similarity would be calculated as:

\[
\sqrt{0.5(0.4 - 0.3)^2 + 0.5(1 - 1)^2} = 0.0707
\]

If this value is the best similarity seen so far, the best location for \( A \) is set at \( A \)'s current position \((1,1)\).

Figure 6.2: The active user \( A \) is attracted to \( B \) who is both \( A \)'s best friend and neighbourhood best

If \( A \) and \( B \) subsequently move in the space, \( A \) will be attracted to the current position of \( B \) and also to his best location, see figure 6.3 below.

Figure 6.3: \( A \) is attracted to both \( B \) and his best location
6.1.3. Experiments with Modified ClusterPSO

The experiments in this chapter are the same as those outlined in 5.3. Again, users are running in parallel and the results for each run are obtained simultaneously for the 10 and 50 active users, for experiments 1 and 2 respectively.

ClusterPSO vs. Modified ClusterPSO

The initial experiment to observe the movement of the users was carried out. The parameter values that were found to be effective for the earlier work were used. These are shown in table 6.1 below.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of users</td>
<td>10/50</td>
</tr>
<tr>
<td>(The number of users in the swarming space at each iteration)</td>
<td></td>
</tr>
<tr>
<td>maximum number of iterations for each run</td>
<td>1000</td>
</tr>
<tr>
<td>(The iteration at which the current neighbourhood for each active user is used to give recommendations)</td>
<td></td>
</tr>
<tr>
<td>swarming space size</td>
<td>1500/7500</td>
</tr>
<tr>
<td>(The width and length of the swarming space)</td>
<td></td>
</tr>
<tr>
<td>maximum range of neighbourhood</td>
<td>150</td>
</tr>
<tr>
<td>(The radius around the active user which is considered to be the user’s neighbourhood)</td>
<td></td>
</tr>
<tr>
<td>maximum velocity</td>
<td>50</td>
</tr>
<tr>
<td>(The maximum velocity allowed at each iteration)</td>
<td></td>
</tr>
<tr>
<td>repulsive factor</td>
<td>1</td>
</tr>
<tr>
<td>(A variable which controls the effect of a repulsive force)</td>
<td></td>
</tr>
<tr>
<td>number of runs</td>
<td>30</td>
</tr>
<tr>
<td>(The number of times the system was run)</td>
<td></td>
</tr>
</tbody>
</table>

Table 6.1: Experimental Parameters

The following figure 6.4 shows the results obtained from experiment 1 compared to those obtained from the original ClusterPSO system with same parameter values. Because the number of users for this experiment is 10, the swarming space size of 1500 was used.
The performance of the two systems for most active users is similar. This shows that the new swarming space and best location calculation have not had a detrimental effect on the clustering abilities of the algorithm – a necessary finding before the feature weights are “activated” in the full system.

However, by looking at the visualisation of the results obtained by the modified ClusterPSO, it was found that users were constantly moving without forming fixed neighbourhoods. A screenshot of the final iteration of a run is shown below in figure 6.5. It can be seen that the users were dispersed around the space.

The reason for this was that there was now an additional best location term to be taken into account. For most active users, it seemed that they were oscillating between their
best friend and the best location whilst repelling the outsiders. Because in many cases each user had different weights (best location), it was not always the case that the users were each other's best friend. The ripple effect of users chasing each other was observed and constant movement was seen throughout as shown below in figure 6.6 where user $A$ was attracted to his best friend, $B$ and his best location whilst user $B$ was attracted to his own best location and his best friend, $C$. In addition to this, the limited space meant that the users were forced to interact with each other and this could inevitably include outsiders which caused a repulsive force for the active users and hence more unavoidable movement.

![Figure 6.6: Ripple effect of users chasing after one another](image)

**Swarming Space Size**

In an attempt to reduce oscillations between users and increase the stability in the system, the swarming space size was increased to 7500. This value was shown to be effective for the original ClusterPSO. The other values were kept the same as in the previous experiment, see table 6.1. This is thought to give users enough space to be close to users that are similar whilst at the same time avoid those that are not similar.

The results from experiment 1 with 10 users exploring a larger swarming space show that the overall performance of the modified ClusterPSO has improved. Figure 6.7 below illustrates this.
Looking further into the visualisation of the system dynamics, it was noticed that stability was achieved where less oscillations were seen in most cases. A screenshot displaying this is shown below in figure 6.8. It can be seen that the users were attracted more towards each other forming clusters. However, in many runs, it was observed that most users moved towards the bottom left corner which was the default best location for users who had not come across any other users in the system. This was confirmed by the best friend list where in many runs, a number of users ended up without any best friends. The runs which resulted in the highest performance were those for which all users found their best friend, it meant that this space size worked well but the maximum range would need to be increased in order for users to have a higher chance of interacting with each other.

Figure 6.8: Visualisation of modified ClusterPSO with swarming space = 7500
Number of Users

Before looking at the maximum range, an obvious way to increase interactions between users is to increase the number of users in the system. By doing so, the number of runs with users having no best friends should decrease.

The number of users was increased to 50 in the following experiment with other parameter values remaining the same.

The results obtained from this experiment with 50 users were compared to those obtained previously with 10 users, see figure 6.9 above. When the number of users was increased, the system performed equally well (or better) for 9 out of 10 users.

From the bestfriend list, it was found that in experiment 2, all users had at least one best friend in most runs. However, the final neighbourhood for most users was very small, in most cases, consisting of only 1 user i.e. the best friend.

From the visualisation of the dynamics, it was seen that the users still had a tendency to move towards the default best location because no other user was encountered. Because the best location was updated or calculated when a user encountered another user(s), it meant that in most cases the first encounter with other user(s) usually happened once the user was close to the default location. This therefore meant that most users will have their best friend who was also located in the same area and hence, the benefits of the new ‘best location’ attractor were minimised, see figure 6.10 below.
6.2. ClusterWeight – full system with activation of feature weights

Thinking back to earlier work on the GA and PSO recommender systems in chapters 3 and 4 respectively, it was learnt that making use of feature weights to represent the user preferences proved to be successful in improving the accuracy of the final predictions and hence recommendations. However, the process of finding a set of weights for each individual user using these two algorithms took a relatively long time especially if there were many users in the system. Because of this, ClusterPSO was introduced which allows a large number of users to be in the system with users themselves being responsible for finding their own neighbourhood. This is a fast technique that requires less computation as each user only needs to calculate a similarity value for users that are within range, rather than for all users in the system at every iteration. Nevertheless, it was discovered that the similarity function used did not accurately evaluate the similarity between 2 users. To solve this problem, the idea of feature weights was reintroduced into the modified ClusterPSO in the form of a new swarming space and a new ‘best location’ attractor, used by the similarity function. ClusterWeight builds on these modifications to ClusterPSO, and uses the position of a user in the space to define feature weights for recommendations in addition to similarity measurements used for clustering. Thus, users now simultaneously search for their user preferences and neighbourhood. Figure 6.11 below describes the ClusterWeight algorithm.
LOOP 1 for each active user, A, in current Profile Selection set
  Create profile(A)
  Initialise a random initial position for A
  Initialise best location, $X_{best}$, to A's initial position
  Create empty best friend and outsider lists for A
END LOOP 1

LOOP 2
  LOOP 3 for each active user, A, in current Profile Selection set
  Create an empty neighbourhood for A
  Reset nbest and outsider list for A
  Add users within range of A to neighbourhood
  Map A's current position to a set of feature weights, $w$
  if A's neighbourhood is not empty
    LOOP 4 for each user j in A's neighbourhood
      Compute similarity value between A and j, $eucdist(A,j)$ with w
      if similarity value is greater than similarity threshold
        Remove j from neighbourhood
      add j to A's outsider list
    END LOOP 4
  END LOOP 3
  LOOP 5 for each item i in training set for A
    Compute the predicted rating for i, predict_rating(A,i), with A's neighbourhood
  END LOOP 5
  Set fitness score of current position to be the average difference between predicted and actual ratings
  Set A's $nbest$ to be user in A's neighbourhood that is most similar to A
  Set A's $X_{best}$ in Rule 1 to be $nbest$'s current position
  if A's best friend list is empty
    Add nbest to best friend list
    Set A's $X_{best}$ in Rule 1 to be $nbest$'s current position
    Set $X_{out}^j$ in Rule 1 to A's current position
  else
    if nbest is more similar than A's current best friend
      Replace A's current best friend with nbest
      Set A's $X_{best}$ in Rule 1 to be nbest's current position
    if fitness score of current position is better than that of $X_{out}^j$
      Set $X_{out}^j$ in Rule 1 to be A's current position
      if A's outsider list is not empty
        Set A's $X_{out}^j$ in Rule 2 to be the central position of all users in the list
      Set repel to true
    else
      Set repel to false
  END LOOP 6
  LOOP 7 for each item i in training set for A
    Compute the predicted rating for i, predict_rating(A,i), using A's mean rating
  END LOOP 7
  Set fitness score of current position to be the average difference between predicted and actual ratings
  if A's best friend list is empty
    Set A's $X_{best}$ in Rule 1 to be the central position of all users in the system
  if fitness score of current position is better than that of $X_{out}^j$
    Set $X_{out}^j$ in Rule 1 to be A's current position
  if repel is true
    Compute velocity according to Rule 1
    if repel is true
      Update velocity according to Rule 2
      Check velocity limit according to Rule 3
      Update current position of A according to Rule 4
      Reset nbest and outsider list for A
  END LOOP 8
  Increment the number of iterations by 1
UNTIL maximum iteration is reached (END LOOP 2)

LOOP 9 for each active user, A, in current Profile Selection set
  Create an empty neighbourhood for A
  Add users within range of A to neighbourhood
  Map A's $X_{out}^j$ to a set of feature weights, $w$
  if A's neighbourhood is not empty
    LOOP 8 for each user j in A's neighbourhood
      Compute similarity value between A and j, $eucdist(A,j)$, with $w$
      if similarity value is greater than similarity threshold
        Remove j from neighbourhood
    END LOOP 8
  END LOOP 7
  LOOP 10 for each item i in test set for A
    if A's neighbourhood is not empty
      Compute the predicted rating for i using A's final neighbourhood
    else
      Compute the predicted rating for i using A's mean rating
      if the difference between predicted and actual ratings for i is 0
        Increment the number of correct predictions (zero tolerance)
      if the difference between predicted and actual ratings for i is either 1 or 0
        Increment the number of correct predictions (at-most-one tolerance)
  END LOOP 10

Figure 6.11: ClusterWeight algorithm
For the ClusterWeight algorithm in figure 6.11 above, an additional task of evaluating the fitness of the current position of each active user was added to the modified ClusterPSO in figure 6.1 (see loop 5 from line 20 to line 22 and loop 6 from line 43 to line 45). The final best location (converted to a set of feature weights) of each active user represents the user's recommendation preferences which are used to select the final neighbourhood of users to compute predicted ratings for test items (see line 69).

The similarity measure for the ClusterWeight remains the same as that used for the modified ClusterPSO. If users are within range of the active user and the similarity value falls below a similarity threshold value, they are considered to be in the neighbourhood. The current location of the active user is converted into a set of feature weights and used in the computation of similarity between the active user and a user, \( j \).

As mentioned above, the only difference between ClusterWeight and the modified ClusterPSO is that the best location for each active user in the ClusterWeight algorithm is no longer determined by the highest similarity value between the active user and his best friend. Instead, ClusterWeight evaluates the fitness of the current position by computing the average difference between the predicted and actual ratings given by the current neighbourhood of users. This calculation occurs at each iteration of a run and if a better fitness score is encountered, the best location is updated accordingly. An advantage of this system is that even if there is no user within range of the active user, the mean rating of the active user can be used as a default predicted rating and the fitness score can be assigned without having to choose a particular location as default.

### 6.3. Experiments

Two system variables (maximum range and maximum velocity) were evaluated to assess their effect on system performance. The parameter values were kept the same as those outlined in table 6.1 (unless otherwise specified). The purpose of the experiments was to obtain a set of values most suitable for each of the two system variables that would maximise the performance of the ClusterWeight recommender system.

Note that full experimental results and analysis are shown in Appendix C. From the experiments, these results were found:
• **Maximum range**: due to the increase in the number of dimensions of the swarming space from 2 to 4, it was necessary to increase the maximum range value accordingly. For experiment 1 with 10 users, the results obtained from setting the maximum range to 1000 were similar to those when set to 2000. However, further analysis using the 3D visualisation tool revealed that when the maximum range of 1000 was used, there were a few cases where the active user was unable to find a neighbourhood of friends as the users in the system were widely dispersed in the swarming space. This problem was resolved when the maximum range was increased to 2000. The larger value of 2000 for the maximum range remained effective in experiment 2 when the number of users was increased to 50.

• **Maximum velocity**: for experiment 1 with 10 users, the best results obtained from setting the maximum velocity to 200 were similar to those when set to 400. From examining the final neighbourhoods, it was discovered that users in the final neighbourhoods obtained for best runs when maximum velocity was set 200 were usually subsets of those obtained in best runs for the maximum velocity of 400. However, from the visualisation of user dynamics, many runs from the larger maximum velocity of 400 experienced premature convergence where most users formed a single neighbourhood. This problem did not occur when the maximum velocity was set to 200 for experiment 1. For experiment 2 with 50 users, it was found that the users were able to cluster themselves more effectively with the maximum velocity of 400 than with that of 200. Due to the increase in the number of users, the problem of premature convergence was reduced as there was a higher chance for each user to meet and find more similar users and thus, these forces of attraction make users explore the swarming space more.

• From the visualisation of best locations, it was discovered that the active users that were most similar (each other’s best friend), also had similar recommendation preferences i.e. similar feature weights.

So far in this chapter, only the zero tolerance mode of comparing the actual and predicted ratings has been used (in order to be counted as a correct prediction, the actual rating of a test item has to match the predicted rating). However, as mentioned throughout this thesis, decision making is very complicated and volatility surrounding it has to be taken into account and higher leniency has to be made. In order to do this, at-most-1 tolerance level
was introduced where the predicted rating can be counted as correct if its difference from the actual rating is 1 or 0.

The following figures show the results obtained from the ClusterWeight system for experiments 1 and 2 based on at-most-1 tolerance. These are best results obtained using the most suitable parameter values found earlier (the maximum range of 2000 with the maximum velocity of 200 for experiment 1 and the maximum range of 2000 with the maximum velocity of 400 for experiment 2).

From the graphs above, it can be seen that ClusterWeight performed significantly well in at-most-one tolerance mode. By allowing higher tolerance to predicted ratings, the performance for experiment 1, see figure 6.12, reached over 80% accuracy on 8 out of 10 active users, with user 10 achieving 100%. In experiment 2, 38 out of 50 users achieved over 80% with 6 obtaining 100% accuracy, see figure 6.13. This therefore shows that
ClusterWeight is able to compute predictions and hence recommendations at a high level of accuracy.

6.4. Comparison with Pearson Algorithm

The Pearson algorithm described in chapter 3 has been widely used for classical recommender systems under collaborative filtering. It computes correlation values between users, with those most similar to the active user being chosen to provide recommendations.

The following figures compare the results obtained from the Pearson algorithm and the ClusterWeight recommender systems on 3 analysis modes: best, worst and average. The same users and movie items were used for both systems. Each system was performed with the parameter values that were found to be most effective. The two experiments that were conducted differ in the number of users used, experiment 1 has 10 users and the number of users was increased to 50 for experiment 2. Note that the results for the Pearson algorithm were always the same when the same parameter values were used.

![Comparison between Pearson Algorithm and ClusterWeight](image)

Figure 6.14: Comparison between Pearson Algorithm and ClusterWeight - best results for both algorithms for experiment 1 with zero tolerance
Figure 6.15: Comparison between Pearson Algorithm and ClusterWeight - best results for both algorithms for experiment 2 with zero tolerance.

Figure 6.16: Comparison between Pearson Algorithm and ClusterWeight - worst results for both algorithms for experiment 1 with zero tolerance.

Figure 6.17: Comparison between Pearson Algorithm and ClusterWeight - worst results for both algorithms for experiment 2 with zero tolerance.
Figure 6.18: Comparison between Pearson Algorithm and ClusterWeight - average results for both algorithms for experiment 1 with zero tolerance

Figure 6.19: Comparison between Pearson Algorithm and ClusterWeight - average results for both algorithms for experiment 2 with zero tolerance

Figure 6.14 shows that in experiment 1, the best results of the ClusterWeight recommender system outperformed those computed by the Pearson algorithm on all 10 users.

In experiment 2, see figure 6.15, the accuracy for ClusterWeight fell below that of the Pearson algorithm for 5 out of 50 users. For the remaining users, the ClusterWeight performed better – in some cases, accuracy improvements were found to be as great as 24%.

Figure 6.16 shows, for experiment 1, when worst results were considered, the Pearson Algorithm performed as well as (or better) for 7 out of 10 users. However, the ClusterWeight recommender still provided better predictions for 3 users.
In experiment 2, see figure 6.17, the worst results for the ClusterWeight algorithm were worse than those obtained from the Pearson algorithm for all but 5 users.

The average results for experiment 1, shown by figure 6.18, indicated that ClusterWeight performed better than the Pearson algorithm on 5 out of 10 users.

Figures 6.19 shows that the average accuracy of ClusterWeight improved as the number of users increased in experiment 2, with 28 out of 50 users obtaining better predictions.

6.4.1. Analysis of Results

The results in figures 6.14 and 6.15 indicated that ClusterWeight performed better than the Pearson Algorithm on best results. This was expected as the Pearson algorithm only uses one feature, rating, in the similarity measure. ClusterWeight, however, takes into account the active user's feature weights which represent the user recommendation preferences and thus, the predictions or recommendations made by the system can be tailored to suit each individual user. In the case that the user would like to be recommended only by people who have similar taste in movies i.e. only use ratings to calculate similarity as in the Pearson Algorithm, the feature weights for this user would have 1.0 as the weight for ratings and 0.0 for the rest of the features and the ClusterWeight would then act like the Pearson algorithm and use the standard correlation function to measure similarity.

From the worst results, see figures 6.16 and 6.17, it was seen that the ClusterWeight can sometimes obtained low prediction accuracy. This was due to the random factors in the system which could introduce noise into the system dynamics. Moreover, the earlier experiments with various parameter values showed that there were many parameters which affected the performance of the ClusterWeight system and they had to be adjusted accordingly in order for the best possible neighbourhoods of users to be formed. The Pearson algorithm, on the other hand, always gave the same results given the same input parameters. This is one of the advantages that the Pearson Algorithm has over the ClusterWeight algorithm. However, the Pearson algorithm is static and cannot adapt to any changes in the user’s taste. An adjustment or addition to a user’s ratings requires the system to recalculate his correlation values with the rest of the users in the system. A
typical run by the ClusterWeight system took approximately 7 and 42 seconds for 10 and 50 active users respectively. Although the speed of the ClusterWeight system was slightly slower than that of the PA system (which took approximately 3 seconds for a typical run of 10 users and 20 seconds for a typical run of 50 users), the ClusterWeight recommender is adaptive and any adjustment or addition has an immediate effect on the calculation. The average results also suggest that most of the time ClusterWeight still obtained better results than the Pearson Algorithm, see figures 6.18 and 6.19.

As mentioned earlier, in an ideal world, the best possible profiles would be chosen from the entire database. This is not always feasible if the database is large. However, because the task of selecting profiles is performed by each user, the similarity calculation in the ClusterWeight system is only required for those in a user’s neighbourhood. A comparison of computation load of the two systems is shown in an example below.

There are 100 users in the system:

- Pearson Algorithm - for simplicity, we do not take into account the symmetric relation between 2 users i.e. $\text{correlation}(\text{user}_1, \text{user}_2)$ is equal to $\text{correlation}(\text{user}_2, \text{user}_1)$. To compute similarity values between the active user and every other user in the system:
  For each active user, $100-1 = 99$ correlation values have to be found and hence, for 100 active users, the system has to run the correlation function 9900 times.

- ClusterWeight
  In the worst case where all users are in the same neighbourhood, the system has to run the similarity function the same amount of times as the Pearson i.e. 9900 times. This scenario is unlikely to occur due to the repulsive force that separates dissimilar users. On the other hand, if all the users are on their own and do not have other users in their neighbourhood then the system does not require any computation.

From the above, this suggests that the ClusterWeight system is able to cope with a higher number of users. Moreover, the computation task can be distributed and performed in parallel to spread the load across a number of machines.
6.5. Comparison with Genetic Algorithm

The genetic algorithm recommender system was described in detail in chapter 3. This was the first system that introduced the notion of feature weights. These weights are captured and fine-tuned to reflect each user’s preference using a genetic algorithm.

The results of the GA recommender below are those that were used to compare against the PSO recommender in chapter 4.

![Comparison between GA and ClusterWeight - best results for both algorithms for experiment 1 with zero tolerance](image1.png)

**Figure 6.20:** Comparison between GA and ClusterWeight - best results for both algorithms for experiment 1 with zero tolerance

![Comparison between GA and ClusterWeight - best results for both algorithms for experiment 2 with zero tolerance](image2.png)

**Figure 6.21:** Comparison between GA and ClusterWeight - best results for both algorithms for experiment 2 with zero tolerance
Figure 6.22: Comparison between GA and ClusterWeight - worst results for both algorithms for experiment 1 with zero tolerance

Figure 6.23: Comparison between GA and ClusterWeight - worst results for both algorithms for experiment 2 with zero tolerance

Figure 6.24: Comparison between GA and ClusterWeight - average results for both algorithms for experiment 1 with zero tolerance
Figure 6.20 shows that the best results of the ClusterWeight recommender system in experiment 1 outperformed those obtained from the GA recommender on 8 users. The difference in prediction accuracy between the two systems for the remaining two users was small.

The prediction accuracy for the ClusterWeight system improved considerably when the number of users increased and was significantly better for all but 6 active users, see figure 6.21.

Figure 6.22 shows that in experiment 1 the two systems did not outperform one another when the worst results were considered. This was due to random factors in both algorithms which could introduce noise into the system.

In experiment 2, see figure 6.23, the worst results for the ClusterWeight algorithm were worse than those obtained from the GA recommender for all but 19 users. However, in one case, the active user 50 received no correct predictions from the GA system whilst the ClusterWeight produced a slight improvement.

The average results in experiment 1, see figure 6.24, show that the ClusterWeight algorithm outperformed the GA system on 7 out of 10 users.

Figure 6.25 shows that in experiment 2, the average results of the ClusterWeight algorithm were better than those obtained from the GA recommender for 27 out of 50 users.
6.5.1. Analysis of Results

The results in figures 6.20 and 6.21 indicate that the ClusterWeight system performed better than the GA recommender for most users. The GA system was run sequentially for each active user. Only the training movie items of the current active user and all rated items for the rest of the users were used to guide the weight evolution. This was, however, different from the way in which the ClusterWeight was performed. Because active users were run in parallel, only the training movies of all users were used. This meant that had the ClusterWeight been performed sequentially with one active user and the rest of the users were allowed all movie items (not just training items) in the training stage, the predictions obtained would have improved further.

The worst results, shown in figures 6.22 and 6.23 revealed that the overall performance of the GA recommender was higher than that of the ClusterWeight system. This is a trade-off for running all the users at the same time, instead of concentrating on one at a time. Because there is only one solution for each run (the current active user’s evolved feature weights), the GA recommender can work solely on getting the best possible solution. However, in the ClusterPSO system, there are many solutions to be considered and sometimes the system has to compromise. For example, in a case where user 1 is similar to user 2 and user 2 is similar to user 3 but users 1 and 3 are not similar, the ideal solution in ClusterWeight would be to have 1 and 2 near to each other and at the same time, retain distance between 1 and 3. Unfortunately, there may be times where all three users are together, due to the random factors affecting the system dynamics, which in turn affects the system performance and hence, poorer results are obtained. This, however, does not occur often as confirmed by the average results, see 6.24 and 6.25.

By allowing users to find their own neighbourhood, it means that ClusterWeight is a fast and efficient algorithm. Instead of running the system 100 times to find a set of feature weights for 100 active users as in the GA system, the ClusterWeight system is only required to run a single time to achieve the same goal. Similar to the point mentioned earlier in the comparison with the Pearson algorithm, ClusterWeight can allow more users to be considered with less computation load. Whilst the GA recommender has to calculate the similarity value between the current active user and the rest of the users in order to find a neighbourhood of similar users, ClusterWeight only calculates the similarity value
of users that are within range to determine whether to be attracted to or repel them. For 10 active users, a typical run of the GA system took approximately 350 seconds whilst a typical run by the ClusterWeight system took approximately 7 seconds. Similarly, for 50 active users, a typical run by the PSO and the ClusterWeight systems took 8310 and 42 seconds respectively.

The final advantage of the ClusterWeight algorithm over the GA is that in addition to the final set of feature weights obtained, ClusterWeight allows visualisations of the final neighbourhoods of users and the way in which the feature weights were derived to be displayed. This is a self explanatory way to increase understanding of the system and hence increases the users’ faith in suggestions.

6.6. Comparison with PSO Algorithm

A recommender system based on a PSO algorithm was introduced and described in detail in chapter 4. The PSO recommender was shown to perform better than both the Pearson and the GA systems. In addition, compared to the GA system, the PSO recommender was able to obtain results faster. The system uses a number of particles to find the best location on a multi-dimensional space which represents a set of feature weights that best describe the current active user’s recommendation preferences. In chapter 4, variations of the PSO recommender were discussed. Here, the results from the final system, Flocking PSO, are used to compare against those obtained from the ClusterWeight algorithm.
Figure 6.26: Comparison between Flocking PSO and ClusterWeight - best results for both algorithms for experiment 1 with zero tolerance

Figure 6.27: Comparison between Flocking PSO and ClusterWeight - best results for both algorithms for experiment 2 with zero tolerance

Figure 6.28: Comparison between Flocking PSO and ClusterWeight - worst results for both algorithms for experiment 1 with zero tolerance
Figure 6.29: Comparison between Flocking PSO and ClusterWeight - worst results for both algorithms for experiment 2 with zero tolerance

Figure 6.30: Comparison between Flocking PSO and ClusterWeight - average results for both algorithms for experiment 1 with zero tolerance

Figure 6.31: Comparison between Flocking PSO and ClusterWeight - average results for both algorithms for experiment 2 with zero tolerance

Figure 6.26 shows that in experiment 1, both the ClusterWeight and the PSO recommenders performed equally well. However, the accuracy obtained from ClusterWeight was slightly better than those from the PSO recommender on 5 users.

In experiment 2, see figure 6.27, the best results from the ClusterWeight recommender system outperformed those from the PSO algorithm for 38 out of 50 users. Additionally, the accuracy for 3 active users from ClusterWeight was greater than 60%.
Figure 6.28 shows that, the worst results obtained from the ClusterWeight were noticeably worse than those obtained from the PSO recommender on 6 active users.

Figure 6.29 shows that in the second experiment, out of the 50 users the worst accuracy for the ClusterWeight recommender fell below that of the PSO algorithm for 39 active users.

The average performance of the two systems was very similar in experiment 1, see figure 6.30.

Figure 6.31 shows that PSO recommender performed slightly better than the ClusterWeight recommender on average results. In two cases, the difference was as great as 29%.

6.6.1. Analysis of Results

Because the framework of the PSO recommender is similar to that of the GA recommender, most comments that were mentioned about the GA system also apply to the PSO recommender. The best results, see figures 6.26 and 6.27, indicated that the difference in performance between the ClusterWeight and PSO recommenders was small on most cases when the number of users was small. However, when the number of users was increased, the performance of ClusterWeight increased considerably.

Similar to the GA, the PSO recommender was run sequentially for each active user. This meant that, for all users other than the current active user, all rated movie items were used to train the system, whilst the active user only used the training items. Because the ClusterWeight ran all users in parallel, only the training movies for all these users were used. Therefore, had the ClusterWeight been run sequentially for each active user, the performance would have increased further. Moreover, because of the parallelism in the system, the ClusterWeight can achieve the solution for all active users in one run, making the algorithm very fast and efficient. For 10 active users, a typical run of the PSO system took approximately 310 seconds whilst a typical run by the ClusterWeight system took approximately 7 seconds. Similarly, for 50 active users, a typical run by the PSO and the ClusterWeight systems took 7920 and 42 seconds respectively.
As with the PA and GA systems, the PSO recommender system also suffers from the problem of having to use a subset of users from the entire database if the number of users is large. ClusterWeight is designed to handle this problem by using repulsive and attractive forces to allow only users that are similar to be close to each other and hence, reduces computation load. For this reason, the ClusterWeight system is able to cope with a larger number of users in the system.

Finally, the visualisation of the PSO recommender only displays the movements of the particles throughout a run whilst the ClusterWeight shows both feature weights and neighbourhood that are used to provide recommendations. This therefore makes the ClusterWeight easier to comprehend and hence improves the users’ confidence in recommendations. Because all users are running simultaneously, ClusterWeight is ideal for online applications where user profiles are constantly changing. Another improvement to the ClusterWeight which could be easily incorporated would be that, instead of using the difference between the actual and predicted ratings to fine-tune the feature weights, users could participate by providing feedback to the system on how much they like the recommendations generated by the current neighbourhood.

6.7. Comparison with COP_KMEANS Algorithm

In section 5.2.2, a clustering algorithm, based on the k-means algorithm, that also takes into account a set of constraints was employed to cluster users such that dissimilar users cannot be assigned to the same cluster. By running this algorithm, k non-overlapping clusters containing similar users were produced where k was the specified number of clusters. Figures 6.32 to 6.37 compare the best, worst and average results obtained from the COP_KMEANS algorithm against those obtained from the ClusterWeight algorithm.
Figure 6.32: Comparison between COP_KMEANS and ClusterWeight - best results for both algorithms for experiment 1 with zero tolerance

Figure 6.33: Comparison between COP_KMEANS and ClusterWeight - best results for both algorithms for experiment 2 with zero tolerance

Figure 6.34: Comparison between COP_KMEANS and ClusterWeight - worst results for both algorithms for experiment 1 with zero tolerance
Figure 6.35: Comparison between COP_KMEANS and ClusterWeight - worst results for both algorithms for experiment 2 with zero tolerance

Figure 6.36: Comparison between COP_KMEANS and ClusterWeight - average results for both algorithms for experiment 1 with zero tolerance

Figure 6.37: Comparison between COP_KMEANS and ClusterWeight - average results for both algorithms for experiment 2 with zero tolerance

Figure 6.32 shows that with best results in experiment 1, the ClusterWeight outperformed the COP_KMEANS system for 6 out of 10 active users and the two systems performed equally well for the rest of the users.
In experiment 2 with best results, the ClusterWeight outperformed the COP_KMEANS on all active users – in a few cases, the difference was as great as 41%, see figure 6.33.

When worst results were considered, the results obtained from COP_KMEANS were better than those obtained from ClusterWeight for most cases in both experiments 1 and 2, see figures 6.34 and 6.35. The average results for most users were similar for both systems, see figures 6.36 and 6.37.

6.7.1. Analysis of Results

In chapter 5, it was shown that performance between the ClusterPSO and COP_KMEANS was approximately the same for experiment 1. As the number of users increased to 50 in experiment 2, COP_KMEANS failed to assign a number of active users to an existing cluster and as a result, the ClusterPSO outperformed COP_KMEANS for all active users. However, the difference in prediction accuracy was small in most cases.

As the ClusterWeight built upon the modifications to the ClusterPSO, it was expected that the performance of the ClusterWeight would be better than that of the ClusterPSO. This, indeed, was shown from the best results in figures 6.32 and 6.33; not only did the ClusterWeight outperform the COP_KMEANS, the difference in prediction accuracy for most active users was considerably large. Moreover, the speed of the ClusterWeight system was also faster than that of the COP_KMEANS. For 10 active users, a typical run by the COP_KMEANS system took approximately 6 seconds (if stabilised before the maximum number of iterations was reached) and 42 seconds (if the maximum number of iterations was reached) whilst a typical run by the ClusterWeight system took approximately 7 seconds. Similarly, for 50 active users, a typical run by the COP_KMEANS system took approximately 370 seconds (if stabilised) and 690 seconds (if the maximum number of iterations was reached) whilst a typical run by the ClusterWeight system took approximately 42 seconds.

However, from average and worst results, it was seen that the results obtained from COP_KMEANS were more consistent than those obtained from the ClusterWeight. This therefore emphasises the importance of feature weights; bad sets of weights can lead to poor results. As mentioned earlier, because the ClusterWeight system is adaptive; it
continuously learns and adapts to the users’ preferences and thus, the feature weights are constantly fine-tuned and the prediction accuracy can be improved. Similar to the PA system, COP_KMEANS is non-adaptive. This means that whenever changes occur to the user profiles in the system, the existing cluster model would need to be updated. This may require the constraint set to be revised and subsequently, all users to be re-clustered. This process is time-consuming and due to the limitation of the algorithm that each user can only be assigned to a single cluster, there is a high probability that the system will not be able to assign all users to clusters.

6.8. Pilot Study

A pilot study was carried out to assess the quality of recommendations produced by the system using real participants. These users were selected on the basis that they have a personal relationship i.e. friends or family members, with myself so that the results obtained could be understood and analysed further.

From the results, it was found that the representations of some users in the system chose a “best friend” which corresponds to their actual best friend in real life to provide the recommendations. Moreover, the feedback obtained from users confirmed that the recommendations generated were considered to be relevant and useful. Further experiments were performed and demonstrated that by incorporating feedback from users into the system, the prediction accuracy could be improved.

Appendix D provides full experimental results and analysis of the pilot study, together with the system architecture for an online recommender system which will act as guidance for the creation of a commercially viable system.

6.9. Summary

This chapter presented a modified version of ClusterPSO which reintroduced the notion of feature weights into the similarity measure to overcome the problem found with the original ClusterPSO in chapter 5. This was that the standard Euclidean distance function was not accurate enough in computing similarity between two users. In the modified ClusterPSO, an attractor, best location, was added so that each active user would remember the position in the swarming space which returned the greatest similarity value
between himself and his best friend. This position represents the active user’s best location and is constantly updated by the user throughout the run.

From the results, it was found that the modified ClusterPSO was able to match the prediction accuracy of ClusterPSO. This showed that the similarity function with feature weights was suitable for the purpose of evaluating similarity and the new swarming space and best location calculation did not detrimentally affect the clustering abilities of the algorithm – a necessary finding before the feature weights were “activated” in the full system. However, with the modification of user dynamics, it was discovered that users had a tendency to move towards the default location and as a result, the benefits of the new ‘best location’ attractor were reduced.

It was therefore decided that in order to implement a truly personalised recommender system, the feature weights had to be activated for recommendations. The novel algorithm, ClusterWeight, built on these modifications to ClusterPSO, and used the position of a user in the space to define feature weights for recommendations in addition to similarity measurements used for clustering. Thus, users now simultaneously search for their user preferences and neighbourhood.

Experiments demonstrated that ClusterWeight was able to obtain a high level of prediction accuracy. Comparisons between ClusterWeight and the other systems which employed the PA, GA, PSO and COP_KMEANS were made and results showed that, in most cases, ClusterWeight performed the best. Moreover, in ClusterWeight, the similarity computation was only performed for those users that appeared in the active user’s neighbourhood, making this algorithm fast and scalable. Because ClusterWeight was able to deal with a larger number of users, it meant that the need for random sampling was reduced. As the neighbourhood selection process was executed simultaneously for all active users, this made ClusterWeight an ideal candidate for real-time applications.
CHAPTER 7

Conclusions

This chapter presents an overview of the thesis. The hypothesis of this thesis was that capabilities of recommender systems could be improved by the use of evolutionary algorithms and swarm intelligence. The thesis focused on five capabilities and improvements: prediction accuracy, relevant recommendations, user scalability, speed of adaptation and online practicality.

7.1. Review of Thesis Contributions

This section reviews the twelve objectives achieved, together with nineteen contributions (not necessarily in order) made in this work.

Objective 1: Identify crucial components of recommender systems from an in-depth investigation of the literature.

Chapter 2 (section 2.2) investigated all related work in the field of recommender systems. A critical review of different approaches taken by past and current systems was presented. From this, three main problems were identified:

- Slow response time. Most existing techniques require a prediction model (a set of similar users) to be computed for each new user. This can take considerable time to produce, making this inappropriate for real-time systems.
- Not adaptive. Most current systems require their prediction model to be updated or recomputed offline periodically. Between these updates, if there are major changes in the user preferences then the existing model can become inaccurate, resulting in poor recommendations being produced.
- Does not scale well. When the number of users increases, computation time can increase exponentially, especially in correlation-based algorithms where correlation has to be computed for every pair of users in the database.

Thus, three crucial components of recommender systems are response time, adaptivity and scalability.
Objective 2: Investigate how the use of a genetic algorithm (GA) can improve capabilities of recommender systems.

The literature review of recommender systems in section 2.2 highlighted the need for an alternative approach which could dynamically adapt to changes in user preferences. At the same time, the system should be scalable to allow for an increasing number of users and yet remain quick to respond to their requests.

Section 2.3.1 presented Genetic Algorithms as potential candidates since they have these desirable properties. As GAs are adaptive, a genetic algorithm could be used to learn personal preferences of users and provide tailored suggestions which should increase the prediction accuracy of the system and, at the same time, improve the relevance of recommendations. Section 3.2 described the way in which a standard GA could be employed to fine-tune a profile-matching algorithm within a recommender system. Thus, the following contributions were made:

- A representation of user profiles was devised which makes use of multiple features such as demographic information.
- The notion of feature weights was introduced to represent the user preferences. Once evolved or attained, they can explain how recommendations are derived and thus, increase the users' confidence in the system.
- A similarity measure between two users that takes into account these feature weights was proposed. This replaces a more traditional Pearson correlation that only uses a single feature, rating, as the basis for computing similarity.

Moreover, as stated in section 2.3.1, there exist a number of variants of GAs including distributed (Whitley and Starkweather 1990) and parallel (Adeli and Cheng 1994). With these advantages of parallelism and distributed properties, the crucial components of response time and scalability could be satisfied by the use of GAs. In section 3.3.2, it was suggested that the GA recommender system could be run off-line and that the evolved feature weights of the active user could be stored on the user's local machine. A local copy of the system would then be responsible for fine-tuning the weights to suit that user's preferences further. This way the processing load on the server would be reduced and parallelism could be achieved.
Objective 3: Devise a method of calculating the quality of the recommendations in order that a fitness score can be assigned to the corresponding feature weights.

A method of computing a fitness score for a set of feature weights was devised by reformulating the problem of making recommendations into a supervised learning task, enabling fitness scores to be computed by comparing predicted ratings with actual ratings (see section 3.2.4). More precisely, to calculate a fitness measure for an evolved set of weights, the recommender system finds a set of neighbourhood profiles for the active user. 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 ratings of all items in the training set are used as a fitness score. Thus, the following contribution was made:

- A fitness function was devised which reformulates the problem of making recommendations into a supervised learning task, enabling fitness scores to be computed.

Objective 4: Implement a recommender system employing a standard GA to find similar users to provide recommendations and compare its performance (in terms of prediction accuracy) with that of a non-adaptive system.

A recommender system employing an elitist genetic algorithm (see section 3.2) was implemented. The full algorithm is given in figure 3.7.

Section 3.3 outlined four sets of experiments designed to observe the difference in performance between the GA recommender system and a standard, non-adaptive recommender system based on the Pearson algorithm (Breese et al. 1998). Section 3.3.2 presented the comparison in terms of prediction accuracy between the GA and Pearson recommender systems. Experiments demonstrated that, compared to the non-adaptive Pearson algorithm approach, the GA recommender system was able to successfully fine-tune the profile matching algorithm, enabling the system to make more accurate predictions. Thus, the following contribution was made:

- An experimental comparison of the performance of a non-adaptive, Pearson algorithm (PA), and GA recommender systems was made.
Objective 5: Identify advantages and shortcomings of the GA recommender system.

It was shown in section 3.3.2 that the GA recommender system performed better than the Pearson algorithm system in terms of prediction accuracy. Additionally, it was found that the feature weights played a vital role in the accuracy of the system. Wrong choice of weights could result in poor neighbourhood sets, and hence poor recommendations. Furthermore, age was often found to be equally as important as rating, and in some cases more so. It suggested that the theory behind the traditional collaborative filtering, that only rating is required, does not always hold. Thus, the following 2 contributions were made:

- A standard GA was shown to be able to fine-tune a profile-matching task within a recommender system, tailoring it to the preferences of individual users.
- The importance of exploiting features other than rating was shown.

It was, however, observed that the amount of time required for the GA recommender to evolve an initial set of feature weights for an active user could be considerable if the number of users was large, which could result in slow response time.

Objective 6: Investigate how the use of swarm intelligence (SI) can improve capabilities of recommender systems.

Section 2.3.2 presented swarm intelligence, in particular particle swarm optimisation, as another potential candidate which has desirable properties highlighted in the literature review of recommender systems. Similar to GAs, PSO is adaptive and can be used to learn personal preferences of users and provide tailored suggestions. Moreover, from the literature, it was found that the PSO generally reaches an optimal solution faster than GAs, thus making it more appealing as far as response time is concerned. Section 4.1.1 described the way in which the PSO was employed (replacing the GA in chapter 3) to fine-tune a profile-matching process within a recommender system. As this is the first time that a PSO had been used in the field of recommender systems, the following contribution was made:

- Particle swarm optimisation (PSO) was employed in a recommender system for the first time.
Objective 7: Implement a recommender system employing an existing swarm intelligence algorithm and compare its performance (in terms of prediction accuracy and speed of adaptation) with those of the GA and non-adaptive systems.

Flying Geese PSO, a recommender system employing a PSO algorithm (see section 4.1.1) was implemented. The full algorithm is given in figure 4.2. Section 4.1 presented the comparison in terms of prediction accuracy between the Flying Geese PSO, GA and the non-adaptive Pearson algorithm recommender systems. It was found that PSO performed very well compared to the other two systems for all four experiments with both zero and at-most-one tolerance levels. Thus, the following contribution was made:

- An experimental comparison of the performance of the PA, GA and PSO recommender systems was made.

Objective 8: Identify advantages and shortcomings of the SI system.

Experiments (see section 4.1) demonstrated that the Flying Geese PSO system outperformed a non-adaptive approach, namely the Pearson algorithm, and obtained higher prediction accuracy than the GA system in most cases. Moreover, the speed of adaptation of the PSO system was also found to be approximately 10% faster than that of the GA. Thus, the following contributions were made:

- Both adaptive techniques, GA and PSO, were shown to make more accurate predictions than the non-adaptive Pearson algorithm.
- Adaptation was shown to be faster for the PSO than the GA recommender system.

However, from detailed analysis of the results, 2 shortcomings of the Flying Geese PSO system were identified. Firstly, as more users were considered, this sometimes resulted in many less similar users being added to the neighbourhood and hence, lowered the overall prediction accuracy of the system. Secondly, by examining the swarm movements using the 3D visualisation tool, it was discovered that many of the runs did not converge. This suggested that either there were multiple solutions or that the particles were still moving towards a better position.

To overcome the latter shortcoming, a number of variations of the PSO algorithm was presented (see Migrating PSO and Flocking PSO in section 4.2). The Flocking PSO was shown to be the most effective in experiments. Not only was it faster than the Flying
Geese at attaining a solution (when converged), it could take into account all potentially
good areas when multiple solutions existed. This was achieved by updating the particles’
velocity to move towards the central location of all best solutions. Thus, the following
contribution was made:

- A technique to deal with multiple solutions in PSO was demonstrated.

Moreover, a collection of software analysis tools was created to aid the analysis process
which led to six distinct behaviours of PSO dynamics being identified, see section 4.3.
From these behaviours, it would be possible to predict the reliability of results obtained
from a run i.e. good convergence with best solution, no solution or slow convergence
with good solution. Furthermore, the 3D visualisation tool, which was developed to
display the swarm dynamics, could be used to show the users the way in which their final
feature weights were obtained and thus, further increase the users’ trust in the
recommendations. In the course of satisfying this objective, the following contributions
were made:

- Shortcomings of the recommender system using a conventional PSO algorithm
  were identified and variations of PSO algorithm were proposed to overcome
  these limitations.
- Six distinct behaviours of PSO dynamics were identified using new analysis and
  visualisation tools.

Objective 9: Build upon these advantages and shortcomings to devise a novel
adaptive system.

The advantages of the Flocking PSO recommender system were that it was able to learn
the user preferences faster than the GA system and that it provided predictions with a
high level of accuracy. Additionally, the way in which the solution was obtained could be
visualised which should increase the users’ confidence in the system. Nonetheless, it was
found that when the number of users increased, this could sometimes result in many less
similar users being added to the neighbourhood and as a result, the level of prediction
accuracy decreased. Moreover, although the speed of adaptation for the Flocking PSO
was faster that that of the GA, the neighbourhood selection was still performed
sequentially for each active user and could take a substantial amount of time if the
number of users was large.
These advantages and shortcomings led to the creation of a novel algorithm, Cluster PSO, in chapter 5 and subsequently, ClusterWeight in chapter 6 which built upon ClusterPSO. Thus, the following contributions were made:

- A novel algorithm, inspired by real-world scenarios, was described. It allows users to choose their own ‘friends’ to give recommendations. Because all users are moving in parallel, neighbourhood selection for all users can be performed simultaneously and thus, speed of adaptation was increased significantly.

- The notions of ‘best friend’ and ‘outsiders’ were introduced. Best friend is used to describe the user most similar to each active user. Conversely, outsiders are those that are considered dissimilar to each user and who should be avoided. These two terms were shown to be responsible for making the system scalable and removing the need for random sampling.

- A system was demonstrated where each user performs two tasks: find his neighbourhood of friends whilst attaining a set of feature weights that is best at representing his recommendation preferences.

**Objective 10: Compare the behaviour of such a system with an existing non-adaptive algorithm.**

In section 5.2.2, an existing constrained clustering algorithm, COP_KMEANS (Wagstaff et al. 2001), was used to allocate users into $k$ groups such that those that were similar would be assigned to the same group and those that were not would be assigned separately. Experiments demonstrated that, when the number of users was increased, the COP_KMEANS system was unable to completely assign many users to clusters due to the non-overlapping property of clusters. In contrast, it was found in section 5.3 that users in the ClusterPSO system were able to successfully select their own neighbourhood that contained only their most similar users. With the aid of a repulsive force, dissimilar users were automatically excluded. As neighbourhoods for ClusterPSO could overlap, the number of users in the system could be scaled.

**Objective 11: Compare the performance (in terms of prediction accuracy and speed of adaptation) of the new system with those of the previous SI, GA and non-adaptive systems.**
Section 5.3.1 presented the comparison in terms of prediction accuracy between the ClusterPSO, PA, GA and Flocking PSO recommender systems. The final system, ClusterWeight, built on the ClusterPSO with full feature weights activated was presented in chapter 6. Sections 6.4, 6.5, 6.6 and 6.7 compared the ClusterWeight against the PA, GA, PSO and COP_KMEANS recommender systems respectively. It was shown that the ClusterWeight outperformed both non-adaptive systems, PA and COP_KMEANS and performed as good as or better than the GA and Flocking PSO systems. Moreover, the speed of the ClusterWeight algorithm was found to be faster than that of the GA and PSO algorithms, see table 8.1.

Objective 12: Demonstrate that this system can improve capabilities (in particular, relevant recommendations) of recommender systems using a pilot study with real participants.

Section 6.8 described the pilot study in which new users interacted with the full ClusterWeight recommender system. This allowed detailed feedback to be obtained on the quality of the recommendations produced by the system. Section D.1.3 presented the results obtained by the system, showing high prediction accuracy with an average of 87.6% for the at-most-one tolerance level. Feedback received from the users was also shown in section D.1.4, were 13 out of 15 users stated that they agreed with most of the recommendations provided by the system. Additional comments from the users further confirmed that the recommendations were of relevance. Thus, the following contribution was made:

- The system has been assessed in experimental trials and received good feedback from real users.

Section D.1.7 described the system architecture for implementing an online recommender system using the ClusterWeight algorithm. It was proposed that the recommendation engine should take the form of a Java applet, hosted by the server but executing locally on the user’s machine. This would make the system fast and scalable as the processing load required by the system would be distributed. Thus, the following contribution was made:

- The system architecture for an adaptive online recommender system was proposed.
7.2. Review of Capabilities of Recommender Systems

This thesis focused on the improvement of the following 5 capabilities of recommender systems:

- **Prediction Accuracy**: Accuracy improvements were shown throughout the thesis in all experiments with different systems. For example, section 3.3.2 showed the comparison between the PA, GA and PSO systems and sections 6.4, 6.5, 6.6 and 6.7 compared the ClusterWeight against the PA, GA, PSO and COP_KMEANS recommender systems respectively. Results demonstrated that the ClusterWeight algorithm was able to predict with accuracy improvements up to 24% compared to Pearson algorithm (see section 6.4), and up to 41% compared to COP_KMEANS (see section 6.7). Furthermore, the prediction accuracy for 36 out of 37 users in the pilot study was found to be greater than 80% using at-most-one tolerance (see section D.1.3).

- **Relevant Recommendations**: Feedback provided by the users participating in the pilot study in section D.1.4, indicated that most users found the recommendations useful and relevant (see users’ comments in table 7.5). Moreover, 13 out of 15 users also explicitly stated that they agreed with most of recommendations generated by the system (see table 7.4).

- **User Scalability**: In the ClusterWeight system, users were responsible for finding their own neighbourhood of friends. Instead of comparing themselves to every other user in the system, the users only needed to compute a similarity value with those that were within close proximity, see figure 6.11 for the full algorithm. This meant that there was no limit to the number of users in the system (given sufficient computational resources and a swarming space big enough to accommodate all users). Furthermore, the system architecture in section D.1.7 showed that the system could be made more scalable by having the recommendation engine executed locally on the users’ machine.

- **Speed of Adaptation**: In section 4.1, it was observed that the time taken to learn each user’s preferences was less for the Flying Geese PSO than that of the GA system. Rather than learning each user’s preferences sequentially as in the GA and PSO approaches, the ClusterWeight performed this task simultaneously for all users. Moreover, as the ClusterWeight did not compare each user to every other user in the system, the time taken to learn user preferences was considerably less than both GA and PSO systems. Table 7.1 below shows the
time taken by the GA, PSO, ClusterWeight, PA and COP_KMEANS systems to perform a typical run, for 10 and 50 active users.

<table>
<thead>
<tr>
<th>Number of users</th>
<th>Time taken in seconds to perform a typical run (' if maximum number of iterations is reached)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Adaptive systems</td>
</tr>
<tr>
<td></td>
<td>GA</td>
</tr>
<tr>
<td>10</td>
<td>350</td>
</tr>
<tr>
<td>50</td>
<td>8310</td>
</tr>
</tbody>
</table>

Table 7.1: Time taken by the GA, PSO, ClusterWeight, PA and COP_KMEANS systems to perform a run

- **Online Practicality:** This was shown by the system architecture for implementing an online recommender system. By running the recommendation engine on the users’ machine, the system could be made scalable and fast for real-time interactions. In addition, the novel ClusterWeight algorithm enables the system to respond to users’ requests and provide new recommendations in real-time.

The rapid expansion of electronic commerce is threatening to overwhelm us with choices. Information overload makes decisions ever more difficult. Friends and family offer advice and recommendations, but they are not always available or able to help. A computer system that can take into account personal recommendation preferences and continuously fine-tune itself according to them would provide great benefit to consumers and companies alike. This thesis has presented such a system. It has shown, for the first time, that adaptive algorithms can improve recommendations generated by a computer. This work has shown that capabilities of recommender systems can be improved by the use of evolutionary algorithms and swarm intelligence.
7.3. Future Work

This thesis has presented methods in which the capabilities of recommender systems can be improved. Through the course of this thesis, some potential areas for future research have been identified. These are described in this final section.

7.3.1. Improvements to existing ClusterWeight system

Suggestion 1: Feedback Integration

A mechanism to integrate feedback into the system could be produced which would yield better and more consistent results. As seen earlier in section D.1.6, feedback was received in a number of different ways – in some cases, users stated whether they agreed with the recommendations produced by the system, whereas some provided actual ratings for the recommended movies. The latter method of explicitly rating each recommended movie seems both most accurate and convenient as the new information can be incorporated into the user’s profile by the system directly without requiring interpretation. Because ClusterWeight is adaptive, the updated data can be used immediately (i.e. in the next iteration).

Alongside a list of recommendations produced by the system, each user also receives a breakdown of his/her recommendation preferences (in terms of feature weights) and the list of friends which form the neighbourhood that is used to provide recommendations. Receiving feedback on the recommendation preferences and the suitability of these friends could also improve the performance of the system.

A suggestion for incorporating feedback on the user’s recommendation preferences is to have an interactive sliding bar showing the feature weights where the weight for each feature can be adjusted by the user, see figure 7.1 below.
For feedback on the neighbourhood of friends which are used to provide recommendations, a possible approach would be to allow the user to choose whether to keep existing friends, see figure 7.2. Moreover, it should also be possible for the user to add more users to the neighbourhood by choosing from the list of users who have opted for their profile to be made public. Note that the anonymity of users is still preserved as the real name and picture of the users are not required by the system. Allowing users to be involved in selecting their friends makes the process more personal as they will be actively choosing and fine-tuning their own peer group which may change depending on mood or circumstance.

**Suggestion 2: Recommendations from Celebrities**

Taking further this idea of choosing your own friends to be in the neighbourhood, the system can provide “virtual famous friends” – these can be celebrities such as movie stars, singers, football players or even DJs. The user can choose these famous friends (perhaps
those that he/she aspires to be like) to give recommendations based, for example, on film premier that they had attended.

**Suggestion 3: Film Clubs**

Another idea would be to introduce film clubs. These clubs can be based on anything from genres to occupation. For example, the Kung Fu film club, the anime film club or even, the IT professionals’ film club. Users can then choose to subscribe to (or unsubscribe from) these clubs. Once the user has become a member, he can provide recommendations to or receive recommendations from other members in the club.

**Suggestion 4: Interactions with Avatar**

Another possible way to improve the system further is to introduce a “character” to enhance the way in which both items can be recommended to, and feedback received from the customer. This character or *avatar* would be a visual representative of the website with which customers can interact. The rationale behind this is that a user would find it easier to relate to another person or character than a standard web interface. There is then the possibility of a relationship being established between user and avatar. From the point of view of the website owner, such a relationship would encourage increased use of the website and allow more preference data to be captured which would consequently lead to more accurate recommendations. The system can employ the use of a 3D avatar to interact with the user. Moreover, further experiments could be conducted to validate the assumption that users would participate more with the avatar. Figure 7.3 shows an early version of the avatar from collaborative work with Akira Sato, see Appendix E.
Suggestion 5: Secondary Friends

Additionally, the concept of secondary friends, which was mentioned earlier in section D.1.1, could be investigated. Perhaps, an idea of friends introducing more friends could be incorporated into the ClusterWeight algorithm. With a system which has thousands of users, it would certainly be very difficult for all users to ‘meet and interact’. As in real life, one way of extending your circle of friends would be to get to know friends of your existing friends. A suggestion as to how this can be incorporated into the system would be, for each active user $A$ at each iteration, the best friend of $A$’s current best friend is added to the current neighbourhood of $A$ (even if this user is not within range of $A$). Subsequently, $A$ can then decide whether to make this new user, his new best friend or the current neighbourhood best.

Suggestion 6: Confidence Display

So far, the list of recommendations produced for an active user (see Appendix D) includes

- The active user’s recommendation preferences
- Users in the active user’s final neighbourhood that are used to produce recommendations
• Recommended items (only those whose predicted rating is either 4 or 5). Each item is displayed with the predicted rating and the users who rated it along with their rating.

However, a few participants from the pilot study found that the list produced by the system was too long and hence, choosing movies from the list could be difficult. For this reason, another improvement to the system could be to add a display of confidence level for each recommendation produced, i.e. "the system is 99.5% confident that you will like this movie".

7.3.2. Alternative machine learning task and intelligent cluster

The algorithms presented in this thesis can be used to perform general machine learning tasks such as classification or clustering of data. One example of this is spam filtering – the system would be used to learn the preferences of each user and, instead of predicting ratings for movies, it would predict whether an incoming e-mail is considered to be of interest to the user. The training stage would require lists of 'interesting' and 'uninteresting' e-mail to be compiled. Two possible ways that these lists could be obtained are:

• by asking the user to explicitly rate e-mails
• by observing the user's behaviour i.e. the e-mails that are read by the user could be considered to be 'interesting' and those that remain unread considered 'uninteresting'

From these lists, the system would attain (or evolve) a set of feature weights which best describes the user's e-mail preferences where features could be domain names, 'To' field, 'CC' field or keywords. A possible example of feature weights is shown below in figure 7.4. These feature weights would then be used to compute an overall score for each incoming e-mail, indicating the likelihood that the user would find the e-mail of interest.

\[
\begin{array}{cccccc}
@cs.ucl.ac.uk & @spam.com & 'To' & 'CC' & 'conference' & 'chat' \\
0.96 & 0.0 & 0.8 & 0.7 & 0.56 & 0.1 \\
\end{array}
\]

Figure 7.4: Feature weights representing e-mail preferences

According to the feature weights shown in figure 7.4, nearly all e-mail coming from "cs.ucl.ac.uk" would be considered to be interesting. Conversely, those coming from "spam.com" are not wanted by the user as the weight for this domain name is zero. The
third and fourth features reveal that if the user’s e-mail address appears in either the ‘To’ or ‘CC’ fields, the e-mail would gain a higher score. The system would scan the subject and content of the e-mail to check whether any defined keywords (in this case, “conference” and “chat”) are found and adjust the overall score accordingly.

7.3.3. Incorporating lifestyle data

In this thesis, only movie data was considered. In order to understand users fully, we should incorporate other lifestyle activities such as shopping for clothes or eating out, into the system.

As the recommender system presented in this thesis is generic, it can be modified to work with most types of items such as books, CDs or even restaurants. Preliminary work on fashion and sizing recommenders has been carried out, see Appendix E.

7.3.4. Applied Psychology

In order to make good recommendations that are useful and of relevant, it is important to understand the factors affecting how people make their choices when buying certain items. A different set of criteria applies depending on the item being considered, whether it is a house, a book or a company stock. Buying clothes is no exception. If we are able to understand and capture the various criteria that are relevant to clothes shopping, it will allow us to improve the accuracy of our user model.

It is also important to realise that everyone has a different set of standards for choosing clothes, so it is necessary to learn the buying habits and preferences of each of our customers. With this information we can create a more effective recommender system and simulate the "real-world" shopping experience.

There are two areas within the fields of psychology and sociology that are highly relevant to shopping behaviour and what influences people's decisions when buying clothes. Although they focus on different parts of the social makeup of our environment, they are highly interdependent and have strong influences upon each other.
Brand awareness

Brands or labels in fashion have been around for a long time, but their significance has exploded in the last few decades. Originally brand names were associated with quality, finesse and style, such as Christian Dior and Chanel. More recently, the fashion market has opened up and many more companies have taken advantage of this to make names for themselves and capture some market share. The importance and influence of brand names have increased phenomenally through aggressive advertising and the use of the media.

However, the increased competition has caused these clothes retailers to evolve to attract as many customers as possible. Changes in their marketing strategies have moved away from traditional values such as quality of manufacturing, and shifted their focus onto more fashionable ways of appealing to a wider audience.

The use of a brand name may suggest various things about a person. It might provide an indication of the income of the person and social status, as well as the sort of company or peer group with which she or he associates. Conversely, these same factors will influence how the person dresses.

Each clothes retailer tries to project its own individual image, something that distinguishes it from the other products in the shelves. This is done in many ways, such as design, fabric, colours, or price. For example, Gap might be associated with comfort and simplicity. Their clothes are not particularly expensive, and this makes their products accessible to the majority of shoppers. On the other hand, Armani portrays itself as a much more exclusive company, pitched at the upper end of the market.

The large majority of these companies have established themselves as household names across the world due to intense competition in the market. Despite this, the success of each brand in a particular geographical location depends very much on its local presence and publicity.

It may be possible to draw some conclusions on which brands of clothing a particular person is likely to wear based on a combination of his or her nationality, wealth, social status, peer group and the geographical location.
Multi-cultural consumer behaviour

Because the web has global coverage, there is no geographical limit to the user base. A person may log on to a website from Bangkok, as easily as someone in London, or anywhere else in fact. All these users will see the same website and the same content. This has both advantages and disadvantages. Whilst the transgression of racial, social and other barriers is in most cases a good thing, there are certain cases where it is still important to recognise certain differences.

For example, climate is one difference that cannot be ignored. It is unlikely that woolly scarves, gloves and hats will be in high demand in Kenya, in much the same as summer beachwear is not likely to sell particularly well in Alaska.

These local factors are significant and need to be taken into account, especially when we are looking at fashion. Another example is local culture. Oriental people are relatively conservative and this is largely reflected in the way they dress. Dressing in a provocative manner such as tight close-fitting clothes or short garments that reveal a lot of flesh is generally frowned upon in South East Asia.

These are two complex areas that have large degree of inter-dependency. It is clear that they have a great deal of influence on how people shop. Further research and analysis will be required to show exactly how they can contribute the recommendation engine.
Bibliography


243


Appendix A

Standard Deviation Values for the Pearson Algorithm, Genetic Algorithm and Particle Swarm Optimisation Recommender Systems

Tables A.1 to A.4 show the standard deviation values of the results obtained from experiments 1 to 4, respectively, for the PA, GA and PSO systems in chapter 4. Note that the predictions computed with the Pearson Algorithm always remained the same given the same parameter values, thus the standard deviation is zero and not shown in A.1 and A.2.

<table>
<thead>
<tr>
<th>Active User</th>
<th>Standard Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GA</td>
</tr>
<tr>
<td>user 1</td>
<td>4.811862736</td>
</tr>
<tr>
<td>user 2</td>
<td>1.564329389</td>
</tr>
<tr>
<td>user 3</td>
<td>0</td>
</tr>
<tr>
<td>user 4</td>
<td>0</td>
</tr>
<tr>
<td>user 5</td>
<td>0.761124395</td>
</tr>
<tr>
<td>user 6</td>
<td>1.928610999</td>
</tr>
<tr>
<td>user 7</td>
<td>2.876779809</td>
</tr>
<tr>
<td>user 8</td>
<td>0.490132518</td>
</tr>
<tr>
<td>user 9</td>
<td>0.479463301</td>
</tr>
<tr>
<td>user 10</td>
<td>7.560103995</td>
</tr>
</tbody>
</table>

Table A.1: Standard deviations for GA and PSO in experiment 1
### Table A.2: Standard deviations for GA and PSO in experiment 2

<table>
<thead>
<tr>
<th>Active User</th>
<th>Standard Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GA</td>
</tr>
<tr>
<td>user 1</td>
<td>2.669358683</td>
</tr>
<tr>
<td>user 2</td>
<td>0.430183067</td>
</tr>
<tr>
<td>user 3</td>
<td>1.112106834</td>
</tr>
<tr>
<td>user 4</td>
<td>0.305128577</td>
</tr>
<tr>
<td>user 5</td>
<td>1.162636718</td>
</tr>
<tr>
<td>user 6</td>
<td>1.4567933</td>
</tr>
<tr>
<td>user 7</td>
<td>1.68461599</td>
</tr>
<tr>
<td>user 8</td>
<td>0</td>
</tr>
<tr>
<td>user 9</td>
<td>0</td>
</tr>
<tr>
<td>user 10</td>
<td>1.357821108</td>
</tr>
<tr>
<td>user 11</td>
<td>3.52084531</td>
</tr>
<tr>
<td>user 12</td>
<td>0.681445387</td>
</tr>
<tr>
<td>user 13</td>
<td>2.792642221</td>
</tr>
<tr>
<td>user 14</td>
<td>1.533158411</td>
</tr>
<tr>
<td>user 15</td>
<td>1.587853578</td>
</tr>
<tr>
<td>user 16</td>
<td>3.347496837</td>
</tr>
<tr>
<td>user 17</td>
<td>0.787838597</td>
</tr>
<tr>
<td>user 18</td>
<td>0.681445387</td>
</tr>
<tr>
<td>user 19</td>
<td>0.498272879</td>
</tr>
<tr>
<td>user 20</td>
<td>1.16684562</td>
</tr>
<tr>
<td>user 21</td>
<td>2.144495081</td>
</tr>
<tr>
<td>user 22</td>
<td>1.86061876</td>
</tr>
<tr>
<td>user 23</td>
<td>1.042212501</td>
</tr>
<tr>
<td>user 24</td>
<td>1.295882072</td>
</tr>
<tr>
<td>user 25</td>
<td>1.116748153</td>
</tr>
<tr>
<td>user 26</td>
<td>1.104327954</td>
</tr>
<tr>
<td>user 27</td>
<td>0.678910554</td>
</tr>
<tr>
<td>user 28</td>
<td>0.484234198</td>
</tr>
<tr>
<td>user 29</td>
<td>0.6149479</td>
</tr>
<tr>
<td>user 30</td>
<td>0.449776445</td>
</tr>
<tr>
<td>user 31</td>
<td>1.79010578</td>
</tr>
<tr>
<td>user 32</td>
<td>2.035602653</td>
</tr>
<tr>
<td>user 33</td>
<td>0.660894552</td>
</tr>
<tr>
<td>user 34</td>
<td>0.610257133</td>
</tr>
<tr>
<td>user 35</td>
<td>1.23404205</td>
</tr>
<tr>
<td>user 36</td>
<td>0</td>
</tr>
<tr>
<td>user 37</td>
<td>0.758908404</td>
</tr>
<tr>
<td>user 38</td>
<td>1.62062514</td>
</tr>
<tr>
<td>user 39</td>
<td>0.668675135</td>
</tr>
<tr>
<td>user 40</td>
<td>0.490132518</td>
</tr>
<tr>
<td>user 41</td>
<td>3.013570838</td>
</tr>
<tr>
<td>user 42</td>
<td>3.202787222</td>
</tr>
<tr>
<td>user 43</td>
<td>3.476378722</td>
</tr>
<tr>
<td>user 44</td>
<td>2.59615271</td>
</tr>
<tr>
<td>user 45</td>
<td>0.4660916</td>
</tr>
<tr>
<td>user 46</td>
<td>0.770132098</td>
</tr>
<tr>
<td>user 47</td>
<td>0.808716878</td>
</tr>
<tr>
<td>user 48</td>
<td>0.347549504</td>
</tr>
<tr>
<td>user 49</td>
<td>2.600839829</td>
</tr>
<tr>
<td>user 50</td>
<td>0.507416263</td>
</tr>
</tbody>
</table>

### Table A.3: Standard deviations for PA, GA and PSO in experiment 3

<table>
<thead>
<tr>
<th>Active User</th>
<th>Standard Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PA</td>
</tr>
<tr>
<td>user 1</td>
<td>22.5836885496091</td>
</tr>
<tr>
<td>user 2</td>
<td>1.4337195</td>
</tr>
<tr>
<td>user 3</td>
<td>2.14127473</td>
</tr>
<tr>
<td>user 4</td>
<td>1.436790805</td>
</tr>
<tr>
<td>user 5</td>
<td>1.430778263</td>
</tr>
<tr>
<td>user 6</td>
<td>3.616850729</td>
</tr>
<tr>
<td>user 7</td>
<td>7.268654162</td>
</tr>
<tr>
<td>user 8</td>
<td>6.053956168</td>
</tr>
<tr>
<td>user 9</td>
<td>2.723148947</td>
</tr>
<tr>
<td>user 10</td>
<td>2.112088904</td>
</tr>
<tr>
<td>user 11</td>
<td>15.37504322</td>
</tr>
</tbody>
</table>

* duplicate user is not considered
<table>
<thead>
<tr>
<th>Active User</th>
<th>PA</th>
<th>GA</th>
<th>PSO</th>
</tr>
</thead>
<tbody>
<tr>
<td>user 1</td>
<td>25.47401</td>
<td>22.135008939</td>
<td>30.432310988723</td>
</tr>
<tr>
<td>user 2</td>
<td>4.3319595</td>
<td>5.7247259</td>
<td>5.963508</td>
</tr>
<tr>
<td>user 3</td>
<td>2.66975396</td>
<td>2.887945617</td>
<td>3.376729016</td>
</tr>
<tr>
<td>user 4</td>
<td>2.426861177</td>
<td>2.303420745</td>
<td>1.809616712</td>
</tr>
<tr>
<td>user 5</td>
<td>4.18172749</td>
<td>4.998850443</td>
<td>4.77589734</td>
</tr>
<tr>
<td>user 6</td>
<td>4.51816258</td>
<td>5.999774215</td>
<td>3.76149395</td>
</tr>
<tr>
<td>user 7</td>
<td>6.718990072</td>
<td>7.314466357</td>
<td>9.481742892</td>
</tr>
<tr>
<td>user 8</td>
<td>2.180512923</td>
<td>2.9959743</td>
<td>2.18432697</td>
</tr>
<tr>
<td>user 9</td>
<td>1.441981395</td>
<td>1.27936766</td>
<td>1.629117242</td>
</tr>
<tr>
<td>user 10</td>
<td>5.6333363734</td>
<td>8.023600247</td>
<td>5.660944961</td>
</tr>
<tr>
<td>user 11</td>
<td>2.238664042</td>
<td>3.95629476</td>
<td>3.503638549</td>
</tr>
<tr>
<td>user 12</td>
<td>2.437494474</td>
<td>3.194103763</td>
<td>2.809041051</td>
</tr>
<tr>
<td>user 13</td>
<td>8.105908162</td>
<td>10.80150053</td>
<td>6.988085098</td>
</tr>
<tr>
<td>user 14</td>
<td>2.795315758</td>
<td>3.704610362</td>
<td>3.57517647</td>
</tr>
<tr>
<td>user 15</td>
<td>2.459701399</td>
<td>2.51455533</td>
<td>2.517981311</td>
</tr>
<tr>
<td>user 16</td>
<td>7.424717796</td>
<td>6.91675945</td>
<td>3.816027904</td>
</tr>
<tr>
<td>user 17</td>
<td>1.455864084</td>
<td>1.595971941</td>
<td>1.284746945</td>
</tr>
<tr>
<td>user 18</td>
<td>3.319494987</td>
<td>5.459847913</td>
<td>5.208394598</td>
</tr>
<tr>
<td>user 19</td>
<td>1.361387553</td>
<td>1.14721051</td>
<td>1.398274799</td>
</tr>
<tr>
<td>user 20</td>
<td>2.377093753</td>
<td>1.938419785</td>
<td>2.284631312</td>
</tr>
<tr>
<td>user 21</td>
<td>5.60008213</td>
<td>6.124756622</td>
<td>5.91316474</td>
</tr>
<tr>
<td>user 22</td>
<td>3.310902559</td>
<td>3.584930144</td>
<td>3.22151314</td>
</tr>
<tr>
<td>user 23</td>
<td>3.95968828</td>
<td>4.523374795</td>
<td>4.33122336</td>
</tr>
<tr>
<td>user 24</td>
<td>3.05858888</td>
<td>4.06781593</td>
<td>3.168813549</td>
</tr>
<tr>
<td>user 25</td>
<td>3.232655382</td>
<td>4.319961686</td>
<td>3.17841372</td>
</tr>
<tr>
<td>user 26</td>
<td>5.544076263</td>
<td>4.996550534</td>
<td>4.857154931</td>
</tr>
<tr>
<td>user 27</td>
<td>1.7762805</td>
<td>1.156689688</td>
<td>1.978040363</td>
</tr>
<tr>
<td>user 28</td>
<td>3.005322629</td>
<td>2.71245901</td>
<td>2.505167074</td>
</tr>
<tr>
<td>user 29</td>
<td>2.38385733</td>
<td>2.233753606</td>
<td>2.096947856</td>
</tr>
<tr>
<td>user 30</td>
<td>2.459791599</td>
<td>2.341283664</td>
<td>2.282344058</td>
</tr>
<tr>
<td>user 31</td>
<td>1.973677354</td>
<td>1.80961712</td>
<td>1.700576146</td>
</tr>
<tr>
<td>user 32</td>
<td>1.874204812</td>
<td>1.422318004</td>
<td>1.69683307</td>
</tr>
<tr>
<td>user 33</td>
<td>2.237609562</td>
<td>2.161317099</td>
<td>1.86955902</td>
</tr>
<tr>
<td>user 34</td>
<td>1.641768041</td>
<td>1.19778013</td>
<td>1.105881107</td>
</tr>
<tr>
<td>user 35</td>
<td>1.464130514</td>
<td>1.250746903</td>
<td>1.412579094</td>
</tr>
<tr>
<td>user 36</td>
<td>1.465699786</td>
<td>0.884086645</td>
<td>0.846901045</td>
</tr>
<tr>
<td>user 37</td>
<td>2.745947903</td>
<td>2.967447915</td>
<td>2.207875142</td>
</tr>
<tr>
<td>user 38</td>
<td>3.213473366</td>
<td>5.353696757</td>
<td>2.48767047</td>
</tr>
<tr>
<td>user 39</td>
<td>2.38385733</td>
<td>2.233753606</td>
<td>2.096947856</td>
</tr>
<tr>
<td>user 40</td>
<td>2.374432782</td>
<td>1.965214738</td>
<td>2.029665054</td>
</tr>
<tr>
<td>user 41</td>
<td>3.244092775</td>
<td>3.747949631</td>
<td>2.56105363</td>
</tr>
<tr>
<td>user 42</td>
<td>5.707062188</td>
<td>4.912658914</td>
<td>4.037328584</td>
</tr>
<tr>
<td>user 43</td>
<td>5.760048691</td>
<td>8.956151164</td>
<td>8.83760866</td>
</tr>
<tr>
<td>user 44</td>
<td>6.123724357</td>
<td>3.98209584</td>
<td>3.7479463</td>
</tr>
<tr>
<td>user 45</td>
<td>2.27024734</td>
<td>2.585474809</td>
<td>1.88995341</td>
</tr>
<tr>
<td>user 46</td>
<td>1.080028074</td>
<td>1.501340397</td>
<td>1.773366174</td>
</tr>
<tr>
<td>user 47</td>
<td>2.0128895</td>
<td>1.75158225</td>
<td>2.092405334</td>
</tr>
<tr>
<td>user 48</td>
<td>3.334358976</td>
<td>3.266901249</td>
<td>2.98825327</td>
</tr>
<tr>
<td>user 49</td>
<td>4.272477202</td>
<td>4.689083728</td>
<td>4.889034799</td>
</tr>
<tr>
<td>user 50</td>
<td>1.270351567</td>
<td>1.45467933</td>
<td>1.080653399</td>
</tr>
</tbody>
</table>

Table A.4: Standard deviations for PA, GA and PSO in experiment 4
Appendix B

Experiments with 4 system variables for ClusterPSO

Note that parameter values used in the experiments were the same as those in table 4.1 (unless otherwise specified).

B.1. Swarming Space Size

For the initial experiment, both the width and length of the swarming space were set to 1500 units.

The number of users for experiments 1 and 3 was 10 and for experiments 2 and 4, 50 users were used. The results obtained from the four experiments are shown below in figure B.1. To enable a fair comparison, only the results for the first 10 users were considered as these are the 10 common users in all the four experiments.

Figure B.1: Comparison between experiments 1 to 4 - best results with swarming space size of 1500.

It can be seen that when random sampling was used, the prediction accuracy improved for most active users. More precisely, results obtained from experiment 3 were better than those from experiment 1 and similarly, results obtained from experiment 4 were better than those from experiment 2. This confirms the finding from chapters 3 and 4 that by
randomly selecting users, there is a higher chance that the system can find users that are more similar to the active user from which to make predictions.

However, when the number of users was increased to 50 (in experiments 2 and 4), rather than an increase in performance, the prediction accuracy actually worsened for a number of users. For this reason, the visualisation tool was used to observe the behaviour of users in experiments 1 and 2. Figure B.2 below shows a screenshot of a run at iteration 1000 for experiment 1. It can be seen that when there were 10 users in the swarming space, the users grouped themselves as expected from the manual allocation (see figure 5.2 for the user preference table) – \{1,3,4\}, \{2,5\}, \{6,7,8,10\}, \{9\}. This collection of neighbourhoods was generated for most runs in experiment 1.

For experiment 2 with the number of users increased to 50, a problem of overcrowding was observed, see figure B.3. Even though users were able to move around freely, the swarming space was too small to accommodate all 50 users. As a result, users could end up belonging to neighbourhoods in which they should not be. It is therefore paramount that the correct swarming space size is chosen, too small and the space is overcrowded, too large and the users are too widely scattered.
Further experiments were conducted in order to find the optimal size of the swarming space for the system. The following swarming space sizes were investigated for experiment 2 with 50 users. Again, note that only the results obtained for the first 10 active users were compared against those from the original size of 1500. Other parameter values were kept the same.

<table>
<thead>
<tr>
<th>Size</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size 1</td>
<td>(the original size used for the previous experiment)</td>
</tr>
<tr>
<td>Size 2</td>
<td>(twice the size of the original swarming space)</td>
</tr>
<tr>
<td>Size 3</td>
<td>(as size 1 was shown to work well for 10 users which equates to 150 units per user, applying this ratio for 50 users gives 7500)</td>
</tr>
<tr>
<td>Size 4</td>
<td>(adding the size of the original space to size 3)</td>
</tr>
</tbody>
</table>

Table B.1: Various swarming space sizes for experiment 2

The results obtained experiment 2 with different swarming space sizes are shown below in figure B.4 below.
From the graph, although the difference in accuracy was small, it can be seen that size 3 performed better than the other sizes for most active users. Looking at the visualisation for this, see B.5 below, the space was no longer overcrowded with enough room for users to move around. For this reason, it was found that with the swarming space of 7500 units, smaller groups of users were formed with an average of 1 or 2 users per neighbourhood, compared to 2 or 3 with the original size of 1500. With the swarming space set to 1500 units, the list of best friends for each active user revealed that the list was consistent throughout the 30 runs. Moreover, the best friend list also agreed with the best friends table, see table 5.3.

These observations, however, did not appear to apply to the best friend list obtained for size 3 of 7500 units; not only was the list not consistent for the 30 runs, the best friends found were not those that were most similar to the active user. The reason for this was that a smaller space ensured that all users interacted with each other and therefore the most similar user was often found. When the size of the swarming space was increased, users were able to move around without meeting every user in the system. This problem is revisited later when experiments with other variables are carried out.
Recall that it was found earlier (see figure B.1) that by setting the swarming space size to 1500, when the number of users was increased from 10 to 50, the overall prediction accuracy worsened. Here, the results obtained from experiment 2, with the swarming space increased to its most effective size in terms of prediction accuracy (7500 units), are compared against those original results obtained from experiment 1, see figure B.6 below.

By choosing a more suitable swarming space size for experiment 2, the graph shows that the system performance for experiment 2 improved. This now confirms the previous finding that when the number of users was increased, the prediction accuracy also increased.
It was found that by increasing the swarming space from 1500 to 7500 units for experiment 1 (with 10 users), the prediction accuracy remained the same in most cases, see figure B.7 below.

![Figure B.7: Comparison of best results obtained from experiment 1 with 2 swarming space sizes of 1500 and 7500.](image)

**B.2. Maximum Range of Neighbourhood**

The maximum range of neighbourhood defines the radius around the active user which is considered to be the user’s neighbourhood. A constraint applied to this variable is that the maximum range cannot exceed the swarming space size. Moreover, if the variable is set to be the same as the width or length of the swarming space then it means that everyone can see each other and that all users would belong in the same neighbourhood. It is therefore important to set the maximum range to the right proportion of the swarming space in order for good neighbourhoods to be formed.

The four experiments outlined in section 5.3 were carried out with various ranges. The same parameter values used in B.1 above were kept the same. Because the swarming space size of 7500 was shown be most effective, this value was chosen for the experiments in this section. 4 different range values considered are shown in table B.2 below.

<table>
<thead>
<tr>
<th>Range</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(half the original range used for the previous experiment)</td>
<td>50</td>
</tr>
<tr>
<td>2</td>
<td>(the original range)</td>
<td>100</td>
</tr>
<tr>
<td>3</td>
<td>(One and a half times the original range)</td>
<td>150</td>
</tr>
</tbody>
</table>
Table B.2: Various range values

Figures B.8 to B.11 show the results obtained for experiments 1 to 4, respectively. In this section, all 50 active users are displayed for both experiments 2 and 4.

Figure B.8: Comparison of best results obtained from experiment 1 with 4 different ranges.

Figure B.9: Comparison of best results obtained from experiment 2 with 4 different ranges.

Figure B.10: Comparison of best results obtained from experiment 3 with 4 different ranges.
As shown in figure B.8, the results obtained from different range values for experiment 1 were similar. However, the third range value of 150 units obtained the highest prediction accuracy on 8 out of 10 users. Both first and fourth values also worked well, obtaining the highest prediction accuracy on 7 out of 10 users.

Figure B.9 shows that the second and third range values both performed equally well for experiment 2, obtaining the highest prediction accuracy on 35 out of 50 users.

The results from experiment 3, see figure B.10, demonstrates that the first and third values performed equally well.

The second, third and fourth valued were found to work equally well for experiment 4, see figure B.11. They all obtained the highest prediction accuracy on 30 out of 50 users.

From the above, it was found that the maximum range variable did not have much impact on the system performance in terms of prediction accuracy. Nevertheless, the third range of 150 units generally worked well in all four experiments. It was then decided that more investigations into other aspects of the system were needed in order to understand the importance of this variable.

Data obtained from experiments 1 and 2 were recorded and studied using various analysis tools. By looking at the visualisation of user dynamics, a few interesting points were observed. The screenshots of run 12 at iteration 1000 with the 4 range values for experiments 1 and 2 are shown in figures B.12 and B.13 respectively. The top two diagrams are for the first and second ranges and the bottom two show the third and fourth.
range 1 = 50 units
range 2 = 100 units
range 3 = 150 units
range 4 = 200 units

Figure B.12: The visualisation of experiment 1 with different range values for run 12

range 1 = 50 units
range 2 = 100 units
range 3 = 150 units
range 4 = 200 units

Figure B.13: The visualisation of experiment 2 with different range values for run 12
By looking at the dynamics of the users with different ranges, it was discovered that the users were more scattered around the swarming space when the range was small. As the range value increased, larger groups of users were formed. Although, the above screenshots demonstrate only 1 run, this trend was also observed for most runs. Looking into the average number of users belonging to a neighbourhood, as expected the neighbourhood size increased as the range was increased.

Again, from looking at the visualisation of user dynamics, it was discovered that when the range was small, i.e. range 1 of 50 units, there was a great deal of movement for each user. The neighbourhoods did not seem to be stable as users constantly moved from one position to another. This trait was observed in both experiments 1 and 2 but was seen more clearly in the latter. It was also noticed that for this range, some neighbourhoods of users were moving together. This could be due to the fact that, with the small range, the users could only interact with less number of users and therefore there was a high probability that the neighbourhood best was the same user as the active user’s best friend, hence the active user always followed this user. As the range increased, the neighbourhoods became more stable and there was less movement for each user. One explanation for this was that because the maximum range was increased, there was a higher chance of the active user’s best friend being in his neighbourhood and that this best friend’s own best friend (could be the active user himself) was also in the same neighbourhood and hence less movement is required.

Figures B.14 and B.15 below show the average movement of a run from experiments 1 and 2, respectively. At each iteration, the average movement of all the users (i.e. the difference between the current and previous positions) was computed. Because the users were always attracted to their best friend, unless their best friend was in the same neighbourhood and located at the same position, movement would always occur. Thus, average movement was never zero.

As mentioned above, the diagrams below show fluctuations in the average movement for both experiments 1 and 2. Reduced movement meant that most users were happy with their neighbourhood and that most neighbourhoods were stable (i.e. active users’ best friend was in the neighbourhood and no outsiders were present). One reason for the sudden huge increase in the average movement was when one or more outsiders entered a
'stable' neighbourhood and as a result, the users in the neighbourhood became dispersed due to the repulsive forces occurring. It then took several iterations for the users to form a stable neighbourhood again. The magnitude of the average movement was found to be greater in experiment 1 (figures B.14) than in experiment 2 (figure B.15). As there were fewer possible groupings in experiment 1, it was easier for most users to be happy with their current neighbourhood and hence lower levels of average movement. As the number of users was higher in experiment 2, there was a greater chance that the users would discover new best friends and change their neighbourhood more often. Thus, the average movement was consistently higher.

The final observation was that in most cases at earlier iterations, there was little or no fluctuation in the average movement as it continued to drop until a certain point was reached before starting to rise again. This trend was observed in the first two ranges for experiment 1, see figure B.14. This was due to the random initial position of each user being scattered around the swarming space and, when the number of users was small, it was likely that most or all users did not meet any other users in earlier iterations and hence, they continued to move towards the centre. Once the users were at the centre (the lowest average movement level was reached for the first time), they could interact with other users and thus, the attractive and repulsive forces were applied.
Note that the average movement peaked at 50 – this was due to the maximum velocity which limited the distance each user was able to move per iteration. The effect of varying maximum velocity values is investigated in the following section.
B.3. Maximum Velocity

By varying the maximum velocity value, the effect of this variable on the system performance and dynamics can be examined. As shown above, the third maximum range of 150 units was found to work well for all experiments and thus, this value was chosen for the experiment in this section. 3 different maximum velocity values were considered and they are shown in Table B.3 below.

<table>
<thead>
<tr>
<th>Velocity</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Velocity 1</td>
<td>(the original velocity used for the previous experiments)</td>
<td>50</td>
</tr>
<tr>
<td>Velocity 2</td>
<td>(double the original velocity)</td>
<td>100</td>
</tr>
<tr>
<td>Velocity 3</td>
<td>(4 times the original velocity. This allows a user to explore a new region outside the current neighbourhood as the velocity is greater than the maximum range)</td>
<td>200</td>
</tr>
</tbody>
</table>

Table B.3: Various velocity values

Figures B.16 to B.19 show the results obtained for experiments 1 to 4, respectively.

![Figure B.16](image1)

Figure B.16: Comparison of best results obtained from experiment 1 with 3 different velocities.

![Figure B.17](image2)

Figure B.17: Comparison of best results obtained from experiment 2 with 3 different velocities.
In experiment 1, see figure B.16, the results obtained from different maximum velocity values were similar. However, the first velocity was found to be most effective, obtaining the highest prediction accuracy for 9 out of active 10 users.

Figure B.17 shows the results obtained from experiment 2. The first two maximum velocity values were found to work well and the results were similar. However, the second value of 100 units performed slightly better and obtained the highest prediction accuracy for 37 of out 50 users.

Similar to the first experiment, the first maximum velocity value was found to be most effective for experiment 3, see figure B.18.

Results obtained from experiment 4 were similar to those obtained from experiment 2, see figure B.19. Again, the first two maximum velocity values were found to work equally well.
From the results, the first maximum velocity value of 50 units was found to perform well for all 4 experiments. However, when the number of users was increased (in experiments 2 and 4), the performance slightly improved when the maximum velocity was set to 100 units.

The average movement for the three different maximum velocity values was studied and figures B.20 and B.21 illustrate this.

![Average movement for run 2](image)

Figure B.20: Experiment 1 - Average movement for run 2
Figures B.20 and B.21 show the average movement of a run with varying maximum velocity values. As was expected, the overall average movement rose as the velocity value was increased. Additionally, it was noticed that when this value was increased, the movement fluctuated more. This was because when the maximum velocity was larger, the users were able to move as far as they required without having to limit themselves to a short distance.

Using the visualisation of the dynamics, it was found that in experiment 1, the third velocity was quickest in separating users into neighbourhoods (see figure B.22 below). This pattern was observed throughout the 30 runs. This was confirmed by the average movement in figure B.20 that with velocity 3, the average movement reached the first lowest point at iteration 10, compared to iteration 35 with velocity 2 and 45 with velocity 1.
velocity $1 = 50$ units

velocity $2 = 100$ units

velocity $3 = 200$ units

Figure B.22: Experiment 1 – visualisation of run 1

velocity $1 = 50$ units

velocity $2 = 100$ units

velocity $3 = 200$ units

Figure B.23: Experiment 2 – visualisation of run 1
Even though the average movement, see figure B.21, showed similar results for experiment 2, it was not clear from the visualisation in figure B.23 that best neighbourhoods were achieved with the third maximum velocity value. However, by inspecting the neighbourhoods and best friend lists, it was found that the greater the velocity value, the more consistent the neighbourhoods were. More precisely, the neighbourhoods obtained when the third velocity was set were approximately the same in most runs. This was hardly surprising. By increasing the maximum velocity allowed per iteration, the active user was able to move towards his best friend faster. More importantly, increased maximum velocity allowed the user to explore more of the space and hence, interact with more users. Moreover, it was found that, as the maximum velocity increased, users that were most similar to each active user were found in more runs. The best friend list for the third velocity of 200 units was the same as that listed in the manual allocation tables in section 5.2.1 for most of the 30 runs for both experiments 1 and 2.

So far, the assumption has been that if a system is able to find a neighbourhood of users most similar to each active user, the prediction accuracy obtained should be high. ClusterPSO has shown that it has the ability to accomplish this task given appropriate parameter values. However, the results obtained in this section showed that the quality of predictions did not improve as performance of the neighbourhood selection process improved. One reason for this could be that the current similarity function is not accurate; for ClusterPSO, a standard Euclidean distance function has been used for this task. In order to find out, a more accurate way of computing similarity between two users should be adopted. Chapters 3 and 4 employed a modified Euclidean distance function that incorporated the feature weights of each active user in the calculation. As it was shown to work well, this would be the best option to use. However, there is one final variable, the repulsive force, which needs to be investigated.

B.4. Repulsive Factor

As described earlier, the repulsive force is responsible for deterring any outsiders or unwelcome users from entering the neighbourhood. The magnitude of the repulsive force is determined by a repulsive factor. To find out whether the repulsive factor is required in
the algorithm, the following experiments were carried out with three different factor values.

<table>
<thead>
<tr>
<th>Factor</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Factor 1</td>
<td>0</td>
</tr>
<tr>
<td>(no repulsive force)</td>
<td></td>
</tr>
<tr>
<td>Factor 2</td>
<td>1</td>
</tr>
<tr>
<td>(causes the active user to move in the direction opposite to the outsider)</td>
<td></td>
</tr>
<tr>
<td>Factor 3</td>
<td>2</td>
</tr>
<tr>
<td>(has twice the effect as factor 2)</td>
<td></td>
</tr>
</tbody>
</table>

Table B.4: Various repulsive factors

As the maximum velocity of 50 units was shown to be most effective in terms of prediction accuracy for all four experiments, this value was used in the experiments in this section. The results obtained for the four experiments are shown below in figures B.24 to B.27.

![Figure B.24](image1.png)

**Figure B.24:** Comparison of best results obtained from experiment 1 with 3 different repulsive factors.

![Figure B.25](image2.png)

**Figure B.25:** Comparison of best results obtained from experiment 2 with 3 different repulsive factors.
As shown above, the results obtained from the four experiments for this section were very similar, especially in experiments 1 and 3 (with 10 users). This suggests that the repulsive factor did not have any effect on the performance of the system. However, after examining the average movement graphs, see figures B.28 and B.29 below, it was found that the maximum velocity of 50 units limited the effect of the repulsive force. In all three cases, the users were only allowed to move a maximum of 50 units per iteration. It can be seen that when no repulsive force was applied (factor 0), the average movement reached the lowest level (i.e. less movement). In experiment 1, the average movement for factor 0 reached the lowest at 25 units, whereas the lowest level was at 30 when a repulsive factor of 2 was set, see figure B.28. This finding was also true for experiments 2 in figure B.29.
Figure B.28: Experiment 1 - Average movement for run 2
Figures B.30 and B.31 below show the visualisation of user dynamics of a run from experiments 1 and 2, respectively. As expected, the users in factor 0 formed neighbourhoods in the centre as a result of no repulsive force being applied. This was observed in both experiments. As the factor value increased, the users were spread around more and the neighbourhoods became smaller as they only contained users that were similar to each active user.
As mentioned above, due to the maximum velocity being set to 50, the effect of the repulsive force on the system performance could not be accurately observed. The final experiments were therefore carried out with the velocity increased to 200 units. The results obtained for experiments 1 and 2 are shown below in figures B.32 and B.33.

Figure B.32: Comparison of best results obtained from experiment 1 for 3 different repulsive factors with maximum velocity of 200.
From the results above, there was a small improvement in the prediction accuracy when the repulsive force was applied (i.e. factors 1 and 2) in experiment 1, see figure B.32. However, this was not the case for experiment 2, see figure B.33. The results for the three factor values were very similar.

As shown earlier, the repulsive force was required so that only most similar users would appear in the neighbourhood. This was confirmed as the neighbourhoods obtained by the system were once again similar to those that were manually allocated. However, the quality of the predictions obtained by these neighbourhoods did not improve. This again suggests that the similarity function currently used does not give an accurate evaluation of the similarity between two users. For this reason, the similarity function that was employed in the GA and PSO systems should be reintroduced.
Appendix C

Experiments with system variables for ClusterWeight

As described in chapter 6, the idea of a default best location is not used in ClusterWeight. Rather, when no other users are encountered, the fitness score of the user's current position is computed using his mean rating as the predicted rating for all training movies. The initial best location is therefore the active user's initial position and remains so until the user meets other users that can give more accurate predictions. Because the users are also attracted to their best friend and the neighbourhood best, if no users are encountered, the active user continues to move towards the current central position of all users in the system. If this position is far from the best location, the user would take longer to get there due to the additional attraction to the best location.

C.1. Maximum Range

Initial test runs were performed with the parameter values in table 6.1. The results showed that in this system, due to the increase in the size of the swarming space from 2 to 4 dimensions, the maximum range had to be increased significantly in order for users to be able to interact with each other. It was found that the maximum range should not be set to less than 1000 units.

Because the range was increased, the maximum velocity also had to be increased. Again, from the initial test runs, the maximum velocities that were found to work well were greater than 200 units. The following parameter values used for the experiments (unless otherwise specified) are shown in table C.1 below.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of users</td>
<td>10 /50</td>
</tr>
<tr>
<td>maximum number of iterations for each run</td>
<td>1000</td>
</tr>
<tr>
<td>maximum velocity</td>
<td>200</td>
</tr>
<tr>
<td>repulsive factor</td>
<td>2</td>
</tr>
<tr>
<td>number of runs</td>
<td>30</td>
</tr>
</tbody>
</table>

Table C.1: Experimental Parameters
The two maximum range values considered in the following experiments are:

Maximum Range = 1000
Maximum Range = 2000

Figures C.1 and C.2 show the results obtained from experiments 1 and 2 with 10 and 50 users respectively. These results are also compared to the results obtained from using the active user’s mean rating as the predicted rating for all test items. The mean ratings were used when the user did not meet any other users in the system or when the system could not find other users similar to the active user to provide recommendations.

From the above, it was found that the maximum range had little effect on the performance in experiment 1, see figure C.1. However, the results obtained by using the mean rating of the active users were worse than those obtained from the ClusterWeight system. A large
improvement in the prediction accuracy was observed seen in experiment 2, see figure C.2, when the maximum range was increased to 2000 – 42 out of 50 users performed equally well or better. Mean ratings gave the worst predictions on 49 out of 50 users.

As described in chapter 6, users in ClusterWeight perform 2 tasks: find the feature weights (best location) that best describe their preferences and at the same time, find the neighbourhood to provide recommendations. Figure C.3 above shows the screenshots taken from two different runs. A blue square represents the best location for each active user and a red circle displays the active user’s current position (which determines his neighbourhood). It was found that even though most users found their best friend and neighbourhood, there were a few users that were left on their own. It was discovered that, in run 9, users 2, 3 and 5 did not come across any other users throughout the run and thus, their neighbourhood was empty (left diagram of figure C.3). By inspecting the best friend list for run 3, user 10 had no best friend as he did not encounter any other users in the run. This was as expected as it was already mentioned that the initial best position depended on the user’s initial position. If this position was far from the position of other users in the space, it would be more difficult for this user to interact with the other users. Cases like this could be described as the user getting stuck in a local minima and this explained why there were no (or fewer) such cases when the maximum range was increased.
Figure C.4: Visualisation of run 3 for maximum range of 2000 displayed in 2 different angles – 
1st and 2nd features (left) and 3rd and 4th features (right)

Figure C.4 shows two different angles of the swarming space; features 1 and 2 on the left 
and features 3 and 4 on the right. When the maximum range was increased to 2000, all 
users were able to form neighbourhoods. The diagram on the right shows precisely that 
there were 4 separate clusters or neighbourhoods. Even though these neighbourhoods 
were separated ‘physically’, by inspecting the neighbourhood list, it was found that the 
users could actually see other users that belonged to other neighbourhoods due to the 
large maximum range value being set. The visualisation therefore demonstrates that users 
were able to select those that were most similar to them, having ‘interacted’ with other 
users in the system.

Moreover, as expected, increasing the maximum range value meant that the 
neighbourhood size of each active user also increased as users could interact with users 
from a further distance. From looking at the final neighbourhoods obtained for each user, 
it was discovered that for many users, the neighbourhood found with the smaller 
maximum range (1000 units) also appeared as a subset of a bigger neighbourhood 
obtained by increasing the range value to 2000 units. This trend was observed in both 
experiments 1 and 2.

Moreover, it was still the case that when the number of users was increased, the 
performance of the system in terms of prediction accuracy also increased. Figures C.5 and 
C.6 below show the results obtained when the number of users was increased for 
maximum range values of 1000 and 2000 respectively.
Figure C.5: Comparison between experiments 1 and 2 - best results with maximum range of 1000.

Figure C.6: Comparison between experiments 1 and 2 - best results with maximum range of 2000.

Finally, from the visualisation of best locations, it was discovered that the users that were most similar (best friends) also had similar recommendation preferences. The screenshot of the final best locations i.e. final set of weights for each user is shown below in figure C.7.
The best friend list for this run is shown in table C.2 below.

<table>
<thead>
<tr>
<th>Active User</th>
<th>Best friend</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>9</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>9</td>
<td>7</td>
</tr>
<tr>
<td>10</td>
<td>4</td>
</tr>
</tbody>
</table>

Table C.2: Best friend list for run 3

Figure C.8 below shows the final feature weights for users 10 and 4. As they were one another’s best friend, their recommendation preferences were similar. This trend was also observed with other users that were best friends.
C.2. Maximum Velocity

The following experiments were carried out to find the most suitable value for the maximum velocity. As the maximum range of 2000 units was shown to be most effective in the previous experiments, this value was used in this section. The rest of the parameter values remained the same. The two maximum velocity values considered in the experiments are:

Maximum Velocity = 200 (used in the previous experiments)
Maximum Velocity = 400 (double the value for velocity 1)

The following figures C.9 and C.10 show the results obtained from increasing the maximum velocity to 400 compared against those with the maximum velocity set to 200.
From the results, the difference in the prediction accuracy of two different maximum velocity values was small for experiment 1, see figure C.9. Nonetheless, the performance for the larger velocity was found to be slightly better than the one from the previous experiment.

Looking at the final neighbourhood size, it was found that setting the maximum velocity to 400 often resulted in larger neighbourhoods for most users. However, the neighbourhoods obtained from maximum velocity of 200 were usually subsets of those from the maximum velocity of 400. Because the best performance of the two velocity values was similar, the worst and average results were compared. Figures C.11 and C.12 show the comparison of the worst results and figures C.13 and C.14 show that of the average results.

Figure C.10: Comparison between 2 velocity values - best results for experiment 2

Figure C.11: Comparison between 2 velocity values - worst results for experiment 1
In figure C.11, it was seen that the worst results for both systems were similar. However, the smaller velocity performed slightly better on 4 users. When average results were considered, the maximum velocity of 200 performed better on 5 users whilst the results on the rest of the users were similar for both systems.
This was, however, not the case when the number of users was increased to 50 in experiment 2. The two systems performed equally well on both worst and average results, see figures C.12 and C.14.

In order to assess the effect that the maximum velocity has, the visualisation of the user dynamics were carefully studied. An interesting pattern with maximum velocity of 400 emerged. In good runs with 10 users, the final neighbourhoods and feature weights obtained were similar to those obtained with the maximum velocity set to 200. Figure C.15 illustrates an example of a good run with maximum velocity set to 400. The neighbourhoods were similar to those obtained for the maximum velocity of 200, see figure C.4.

![Figure C.15: Visualisation of a good run from velocity 400 (run 1), showing both final neighbourhoods and best locations (left) and best locations alone (right)](image)

Moreover, the runs with poor results were also examined. In experiment 1, it was found that in most cases, with the maximum velocity set to 400, the runs seemed to converge too soon where most users belonged to one large neighbourhood. This was a trade off for having a large maximum velocity. As mentioned earlier, users initially moved towards the central position of all users when they did not encounter any other users in the system. Because the velocity was large, the users took less time to explore the initial area and as a result, most users grouped together in the centre, see figure C.16. This trend did not occur when the maximum velocity of 200 units was used.
In experiment 2 with 50 users, the visualisation revealed that when the maximum velocity was increased to 400, the users formed bigger neighbourhoods. The feature weights found in most good runs also formed clusters of similar users. This again shows that similar users tend to have similar preferences. With the maximum velocity of 200, the users were more scattered and formed smaller neighbourhoods, hence the best locations or feature weights of users were also scattered. Figure C.17 below displays this difference. It was observed that the final neighbourhoods obtained from setting the maximum velocity to 200 still appeared as a subset of those obtained from the maximum velocity of 400. Furthermore, premature convergence cases were found to occur less often with the larger velocity value. Thus, in experiment 1, the maximum velocity should be set to 200 and in experiment 2, the maximum velocity should be increased to 400.
Appendix D

Online Implementation

This appendix focuses on a pilot study in which new users interact with the full ClusterWeight recommender system (users participating in the experiments are friends and family whom I know personally). This allows detailed feedback to be obtained on the quality of the recommendations produced by the system. Moreover, the additional background knowledge of each user allows further analysis of the neighbourhood selection task.

The appendix is organised as follows: section D.1 describes the pilot study carried out with real participants and provides experimental results and analysis. Section D.2 presents potential directions for future work. Finally, section D.3 concludes.

D.1. Pilot Study

The experiments presented in this appendix were carried out to evaluate the performance of the ClusterWeight system (presented in chapter 6) on real data. By collecting data from new users, direct feedback from the users on recommendations generated by the system can be obtained and used to further improve the performance of the system. More importantly, these users were chosen based on my personal relationship with them and relationships amongst themselves. With the background knowledge of each user, results can be analysed further rather than using only statistical methods of evaluations.

D.1.1. Users

A prototype website was set up for this pilot study. 40 people (including myself) were invited to go online and rate movies. Each user was sent their username by email for access to their online account, see figure D.1. After logging on, each user was presented with a list of movies sorted by alphabetical order (Figure D.2). The total number of movies used in this experiment is 1899. In addition to all 1682 movies from the

http://www.swarmrecommend.com/cgi-bin/rate.pl
MovieLens dataset used in previous experiments, 217 recently released (after 1998) movie items were randomly selected and added to make the database more up to date.

Figure D.1: The login page

Figure D.2: The first page after the user has successfully logged on.

Figure D.3: A list of movies beginning with ‘A’. The user can rate a movie by selecting a drop-down list to the right of the movie and choose a rating between 1 (worst) and 5 (best). Clicking on
movie title will open a new window with the movie information from the Internet Movie Database\textsuperscript{10}.

Figure D.4: By clicking on ‘Your rated movies’, this page is displayed which lists the currently rated movies for the user where ratings can be updated.

The users were allowed 2 months to submit new ratings and modify existing ones. After two months, data was collected and used for the experiments in this appendix. However, out of the 40 participants, 3 were unable to provide ratings for the specified minimum number of 40 movies and thus, their ratings were not used. The actual relationships amongst the users can be shown in the diagrams below. Note that the names of the users have been changed for privacy reasons.

Figure D.5 displays the main associations amongst the users. There are two different kinds of associations or relationships considered here: primary and secondary. A primary relationship between two users $x$ and $y$ is defined to mean that $x$ and $y$ are friends (or partners/family members) and have personal knowledge of each other. On the other hand, a secondary relationship between users $x$ and $y$ is defined to mean that both $x$ and $y$ have a primary mutual contact and do not have a primary relationship. Clearly, there is less association between $x$ and $y$ in this case. However, it is possible that the relationship between $x$ and $y$ will change to a primary one as they get to know more about each other as time passes.

\textsuperscript{10} http://us.imdb.com/
There is a relationship between myself aka Supi and each of the users. However, those users towards whom an arrow is pointing represent secondary contacts introduced to myself and the group by a primary contact. With this dataset, there are four main groups amongst the users where each group is shown in a different colour. The same coloured boxes represent users belonging to the same group (and knowing each other). White boxes represent users that do not belong to any existing group. This view of the associations between users is based on personal knowledge.

Figure D.6: User partitions based on age group
Figure D.7: User partitions based on gender

Figure D.8: User partitions based on occupation

Figure D.6 shows user partitions based on different age groups: 25 and below, 26-30, 31-35 and 36 and above. Figure D.7 shows user partitions based on gender: male and female. Figure D.8 shows user partitions based on occupations. However, this view only shows the full-time occupation and does not display some users who can be in more than 1 group e.g. full-time job and also a part-time student.
There are numerous additional relationships amongst the participating set of users and as these are specific to each user, they will be detailed in the analysis section later. In this appendix, we are interested in looking into how social analysis (utilising knowledge between primary contacts) can lead to an improvement of the current system for future work.

D.1.2. Experiments

The ClusterWeight system presented in chapter 6 was employed to provide recommendations. Table D.1 shows the parameter values used to run the experiment. These values were those that had been shown to be most suitable.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of runs</td>
<td>30</td>
</tr>
<tr>
<td>number of iterations</td>
<td>1000</td>
</tr>
<tr>
<td>number of users</td>
<td>37</td>
</tr>
<tr>
<td>swarming space size</td>
<td>7500</td>
</tr>
<tr>
<td>maximum distance neighbourhood search</td>
<td>2000</td>
</tr>
<tr>
<td>maximum velocity</td>
<td>400</td>
</tr>
<tr>
<td>repulsive factor</td>
<td>2</td>
</tr>
<tr>
<td>similarity threshold</td>
<td>0.2</td>
</tr>
</tbody>
</table>

Table D.1: Parameter values

Table D.2 below shows the total number of movie items rated by each user and the breakdown of the training and test items. As explained in section 3.1.2, for each user, 30% of all movie items that the user had rated were randomly selected to be in the training set, with the remaining 70% forming the test set.
<table>
<thead>
<tr>
<th>User_ID</th>
<th>Name</th>
<th>Total movies rated</th>
<th>Training</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Rick</td>
<td>231</td>
<td>69</td>
<td>162</td>
</tr>
<tr>
<td>2</td>
<td>Vanessa</td>
<td>101</td>
<td>30</td>
<td>71</td>
</tr>
<tr>
<td>3</td>
<td>Sabrina</td>
<td>270</td>
<td>81</td>
<td>189</td>
</tr>
<tr>
<td>4</td>
<td>Pamela</td>
<td>400</td>
<td>120</td>
<td>280</td>
</tr>
<tr>
<td>5</td>
<td>Scott</td>
<td>167</td>
<td>50</td>
<td>117</td>
</tr>
<tr>
<td>6</td>
<td>Pippa</td>
<td>318</td>
<td>95</td>
<td>223</td>
</tr>
<tr>
<td>7</td>
<td>Patrick</td>
<td>48</td>
<td>14</td>
<td>34</td>
</tr>
<tr>
<td>8</td>
<td>Pia</td>
<td>46</td>
<td>13</td>
<td>33</td>
</tr>
<tr>
<td>9</td>
<td>Joseph</td>
<td>228</td>
<td>68</td>
<td>160</td>
</tr>
<tr>
<td>10</td>
<td>Brian</td>
<td>430</td>
<td>129</td>
<td>301</td>
</tr>
<tr>
<td>11</td>
<td>Philip</td>
<td>76</td>
<td>22</td>
<td>54</td>
</tr>
<tr>
<td>12</td>
<td>Mark</td>
<td>206</td>
<td>61</td>
<td>145</td>
</tr>
<tr>
<td>13</td>
<td>Bart</td>
<td>252</td>
<td>75</td>
<td>177</td>
</tr>
<tr>
<td>14</td>
<td>Sean</td>
<td>153</td>
<td>45</td>
<td>108</td>
</tr>
<tr>
<td>15</td>
<td>Cilla</td>
<td>79</td>
<td>23</td>
<td>56</td>
</tr>
<tr>
<td>16</td>
<td>Stephen</td>
<td>113</td>
<td>33</td>
<td>80</td>
</tr>
<tr>
<td>17</td>
<td>Thomas</td>
<td>119</td>
<td>35</td>
<td>84</td>
</tr>
<tr>
<td>18</td>
<td>Peter</td>
<td>223</td>
<td>66</td>
<td>157</td>
</tr>
<tr>
<td>19</td>
<td>Julia</td>
<td>191</td>
<td>57</td>
<td>134</td>
</tr>
<tr>
<td>20</td>
<td>Malcolm</td>
<td>97</td>
<td>29</td>
<td>68</td>
</tr>
<tr>
<td>21</td>
<td>Henry</td>
<td>59</td>
<td>17</td>
<td>42</td>
</tr>
<tr>
<td>22</td>
<td>Nigel</td>
<td>51</td>
<td>15</td>
<td>36</td>
</tr>
<tr>
<td>23</td>
<td>Fiona</td>
<td>45</td>
<td>13</td>
<td>32</td>
</tr>
<tr>
<td>24</td>
<td>Sonia</td>
<td>316</td>
<td>94</td>
<td>222</td>
</tr>
<tr>
<td>25</td>
<td>Stacy</td>
<td>140</td>
<td>42</td>
<td>98</td>
</tr>
<tr>
<td>26</td>
<td>Jennifer</td>
<td>145</td>
<td>43</td>
<td>102</td>
</tr>
<tr>
<td>27</td>
<td>Neil</td>
<td>52</td>
<td>15</td>
<td>37</td>
</tr>
<tr>
<td>28</td>
<td>Paddy</td>
<td>294</td>
<td>88</td>
<td>206</td>
</tr>
<tr>
<td>29</td>
<td>Karen</td>
<td>64</td>
<td>19</td>
<td>45</td>
</tr>
<tr>
<td>30</td>
<td>Olivia</td>
<td>75</td>
<td>22</td>
<td>53</td>
</tr>
<tr>
<td>31</td>
<td>Andy</td>
<td>208</td>
<td>62</td>
<td>146</td>
</tr>
<tr>
<td>32</td>
<td>Lee</td>
<td>107</td>
<td>32</td>
<td>75</td>
</tr>
<tr>
<td>33</td>
<td>Yasmin</td>
<td>83</td>
<td>24</td>
<td>59</td>
</tr>
<tr>
<td>34</td>
<td>Timothy</td>
<td>47</td>
<td>14</td>
<td>33</td>
</tr>
<tr>
<td>35</td>
<td>Adam</td>
<td>382</td>
<td>114</td>
<td>268</td>
</tr>
<tr>
<td>36</td>
<td>Edward</td>
<td>288</td>
<td>86</td>
<td>202</td>
</tr>
<tr>
<td>37</td>
<td>Supi</td>
<td>371</td>
<td>111</td>
<td>260</td>
</tr>
</tbody>
</table>

Table D.2: A breakdown of the total number of movie items rated by each user

D.1.3. Results

Figures D.9 to D.10 show the best, average and worst results obtained for each active user with zero and at-most-one tolerance levels, respectively.
Figure D.9 shows that the best run(s) obtained for each user for zero tolerance yielded an accuracy of between 30 to 60%. However, it is important to remember that the reliability of the ratings given by the user is not 100% accurate (illustrated by the experiment carried out by Hill et al. (1995) where users were asked to re-rate the same items after 6 weeks). For this reason, more realistic results are those with at-most-one tolerance level. By using this mode of evaluation, the prediction accuracy of the system for all but 3 active users was above 80%, giving an average of 87.6% for the system performance of best runs, see figure D.10. It can be said that 4 out of 5 movies that are recommended by the system will often be acceptable by the user.
D.1.4. User Feedback

The results shown so far were obtained from the test set of each active user.

To assess the relevance of recommendations, user feedback was required. All users were sent a list of personalised recommendations produced automatically by the system. These recommendations were computed using the final neighbourhood of each active user. Initially, a list of movie items that were rated by at least one user in the active user’s neighbourhood was compiled. All items that had already been rated by the active user were removed from the compiled recommendation list. The predicted rating for each item in the list was then computed using the formula presented in section 5.1.6 where the weight of each user in the neighbourhood (similarity value between the active user and this user) was calculated, using the final set of feature weights from the best run. Only the items that received the predicted rating of either 4 or 5 (good) were recommended to the active user.

The list of recommendations sent to an active user included:

- The active user’s recommendation preference (the set of feature weights attained by the system from the best run)
- Users in the active user’s final neighbourhood that were used to produce the recommendations
- Recommended items (only those whose predicted rating was either 4 or 5). Each item was displayed with the predicted rating and the users who rated it together with their rating

An example list of recommendations (compiled for user Sabrina) is shown below in table D.3. In the first section (recommendation preferences), the number in brackets after each feature represents the weight or the active user’s preference for that feature. In this case, the system concluded that Sabrina preferred to be recommended mostly by people of the same gender and then those in the same age group. In the second section (friends), the similarity value between the active user and his/her friends is presented. In this case, Stacy was considered most similar to Sabrina whilst Patrick was considered least similar. The last section shows the list of recommended movies – the number in brackets next to each movie represents the predicted rating computed by the system for the active user (in this case, Sabrina) and a list of friends recommending that particular movie. The figure in brackets besides each friend represents the actual rating given by that friend for the movie.
your recommendation preferences:
+++ rating (0.21), age (0.27), gender (0.30), occupation (0.22) +++

your friends are:

<table>
<thead>
<tr>
<th>Name</th>
<th>Similarity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stacy</td>
<td>0.8617</td>
</tr>
<tr>
<td>Yasmin</td>
<td>0.8574</td>
</tr>
<tr>
<td>Jennifer</td>
<td>0.5264</td>
</tr>
<tr>
<td>Pippa</td>
<td>0.5247</td>
</tr>
<tr>
<td>Pamela</td>
<td>0.5137</td>
</tr>
<tr>
<td>Karen</td>
<td>0.5060</td>
</tr>
<tr>
<td>Cilla</td>
<td>0.4914</td>
</tr>
<tr>
<td>Sonia</td>
<td>0.4893</td>
</tr>
<tr>
<td>Philip</td>
<td>0.4438</td>
</tr>
<tr>
<td>Lee</td>
<td>0.4362</td>
</tr>
<tr>
<td>Timothy</td>
<td>0.4360</td>
</tr>
<tr>
<td>Rick</td>
<td>0.4353</td>
</tr>
<tr>
<td>Mark</td>
<td>0.4280</td>
</tr>
<tr>
<td>Joseph</td>
<td>0.4278</td>
</tr>
<tr>
<td>Sean</td>
<td>0.4181</td>
</tr>
<tr>
<td>Nigel</td>
<td>0.2768</td>
</tr>
<tr>
<td>Neil</td>
<td>0.26801</td>
</tr>
<tr>
<td>Peter</td>
<td>0.2655</td>
</tr>
<tr>
<td>Adam</td>
<td>0.2584</td>
</tr>
<tr>
<td>Bart</td>
<td>0.2565</td>
</tr>
<tr>
<td>Andy</td>
<td>0.2468</td>
</tr>
<tr>
<td>Patrick</td>
<td>0.2124</td>
</tr>
</tbody>
</table>

Recommendation List for Sabrina

<table>
<thead>
<tr>
<th>Recommended movies</th>
<th>by</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boys on the Side(4)</td>
<td>Pamela(3)</td>
</tr>
<tr>
<td>Mallrats(4)</td>
<td>Joseph(4)</td>
</tr>
<tr>
<td>Desperate Measures(4)</td>
<td>Pippa(4)</td>
</tr>
<tr>
<td>Cop Land(4)</td>
<td>Andy(3)</td>
</tr>
<tr>
<td>Perfect World, A(5)</td>
<td>Joseph(5)</td>
</tr>
<tr>
<td>Executive Decision(4)</td>
<td>Yasmin(3)</td>
</tr>
<tr>
<td>Crash(4)</td>
<td>Adam(3)</td>
</tr>
<tr>
<td>Evita(4)</td>
<td>Pamela(3) Yasmin(4) Bart(3) Rick(5) Andy(2) Jennifer(4)</td>
</tr>
</tbody>
</table>

Table D.3: recommendation list sent to Sabrina

Users were asked to provide feedback on the quality of recommendations by choosing one of three options. 16 out of 37 users replied and the results are shown in table D.4 below.

<table>
<thead>
<tr>
<th>Option</th>
<th>Number of users</th>
</tr>
</thead>
<tbody>
<tr>
<td>agree with most of the recommendations</td>
<td>13</td>
</tr>
<tr>
<td>(i.e. would have themselves given these items a rating of 4 or 5)</td>
<td></td>
</tr>
<tr>
<td>agree with half of the recommendations</td>
<td>2</td>
</tr>
<tr>
<td>agree with less than half of the recommendations</td>
<td>1</td>
</tr>
</tbody>
</table>

Table D.4: User feedback
Some comments received from the users on the overall impression of the system are shown in table D.5.

Edward says “Works very well - obviously it mostly recommends movies that I have yet to view so can't comment.”

Peter says “Out of the films that I've seen, the recommendations are all almost exactly at the rating I would have given them. Very impressive!”

Pippa says “Too much info given. Only want to see a few recommended movies (just the titles)”

Philip says “Lots of films there, I didn't think I liked so many films Mark did!! Most of the films looked like a good match but there were some types of films that kept coming up that that I didn't like, this might have been because I liked a one off in that category and that might have changed the results. There were a lot of films I haven't watched or heard of and I guess this is where your system comes in to use as it gives a selection of films you might like.”

Scott says “there are quite a few I don't like, and a fair few that I haven't seen too... A factor for me is probably that they are all blockbusters, and I tend to go for alternative films…”

Paddy says “I think the majority of the recommendations were fine, but there were some obviously wrong recommendations, e.g. Jackass: The Movie, Transformers: The Movie. If someone had actually recommended them to me, I'd be rather worried!”

Thomas says “I agree with most of the recommendations.”

Mark says “there's a lot I hadn't seen there! Although I could comment on the ones I haven't seen, I expect I'd disagree with quite a few of the recommendations”

Sabrina says “I agree with most of the recommendations (based on those that I have seen or have heard of). Obviously can't say about those that I haven't heard of before”

Rick says “Not a bad recommendation system!!”
**Bart** says “I agree with most but disagree with those that are mostly Disney, kids or teen films. My general impression is there are too many recommendations! I'd rather see 20 or so very definite recommendations, or an ordered list with the most likely hits first. It would take me years to watch all the ones you sent. It is interesting to see who "recommends" each film, just so you can check that it is sane and why you have got it. It makes sense to take recommendations that have a lot of names against them more seriously. I think I'll go and rent the Three Colours trilogy now!!”

**Pamela** says “the movies recommended by Neil and Stacy would be most of the ones I would rate high. The system seems pretty cool! A bit surprised by the range of films though”

**Joseph** says “Good and I will definitely try some of the movies. Some movies are from genres I don't enjoy however, so maybe the genre should play more of a factor?”

**Andy** says “I thought that most of the recommendations were good. Definitely more than half. The presentation is simple and intuitive so, yeah, good job.”

---

**Table D.5: Additional comments given by the users**

---

**D.1.5. Analysis of Results**

From further analysis of the results together with feedback received from users, many interesting observations were found:

- From their best runs, both Pia and Patrick chose one another to be their best friend. For Patrick, his final feature weights attained by the system revealed that he mostly preferred to be recommended by people in the same age group (2nd feature), see figure D.11. Looking at figure D.6 which illustrates user partitions based on age groups, it is found that the other user in the same age group is Pia.
Pia's final feature weights, however, showed that she preferred to be given recommendations by people with the same occupation, see figure D.12 below.

Note also that her feature weight for gender was the lowest out of the four features illustrating that gender was not the main priority for her. Figure D.8 illustrating user partitions based on occupation shows that the only user in the same group as Pia is indeed Patrick. Moreover, the prediction accuracy was found to be similar for both Pia and Patrick, in particular for at-most-one tolerance where the system was able to predict their ratings with 94% accuracy, see figure D.10. In real life, Pia and Patrick have been married for 28 years and this therefore confirms their compatibility with each other. Furthermore, from my personal relationship with them, I can say that they have similar taste in movies.
This was confirmed by their feedback where both users chose the first option which stated that they agreed with most recommendations.

- As mentioned earlier, the recommendation list produced for Sabrina, see table D.3, showed that she preferred to be recommended by people of the same gender. Thus, the first 8 entries in her friends list were female. The visualisation of Sabrina’s best run is shown below in figure D.13 – although difficult to see in the figure, analysis showed that most users surrounding Sabrina were female.

![Swarm recCluster2D](image)

Figure D.13: Final neighbourhoods of Sabrina’s best run

Looking at her rated movies, it was discovered that a high proportion of them belonged to romance and animation genres which are stereotypically liked by women and, for most of these movies, Sabrina gave a high rating of 4 or 5. Figure D.10 shows that the system was able to predict her ratings with an accuracy of 93% with at-most-one tolerance. From her feedback, she too agreed with most recommendations provided by the system.

- It was found that most users who belonged in Philip’s friends list that were used to provide recommendations are also his friends in real life. This was not a surprise as all users belonging to this group in real life were at university together. They are also in the same age group and most of them are working in IT, see

305
figures D.5, D.6 and D.8. Moreover, in all his best runs, Philip’s best friend was found to be Sean and similarly, in Sean’s best runs, his best friend was also Philip. From my personal relationship, Philip and Sean are also closest in real life. The prediction accuracy for Philip shows that the system was able to predict with the accuracy of 89% with at-most-one tolerance and Philip’s feedback confirmed that he too agreed with most recommendations.

- In real life, Edward and Joseph are close friends. The system also discovered this – Edward’s best friend, in all his best runs, was Joseph. However, although Edward appeared in Joseph’s friend list, he was not Joseph’s best friend. Perhaps, a more interesting point was that two other users that consistently appeared in Joseph’s friend list were Neil and Adam. In real life, Neil, Adam and Joseph have a friend in common which is Edward, see figure D.5. Both Edward and Joseph confirmed that they agreed with most recommendations.

- Scott’s feedback, however, showed that he only agreed with half of the recommendations produced by the system. From looking at his list of friends, it was found that the only friend used to provide recommendations for Scott was Vanessa. In my opinion, based on my personal relationship with both of them (who do not know each other in real life), Scott and Vanessa have different taste in movies. Scott’s final feature weights showed that he highly preferred to be recommended by people with similar ratings, see figure D.14 below. It was then discovered that, although the number of movies that Scott and Vanessa had rated in common was small, the ratings that they had given for those common movies were often the same, thus Vanessa was chosen by the system as Scott’s best friend.
Again, in my opinion, Scott is rather particular with his taste in movies and would normally go for independent or art-house movies rather than mainstream. Mark, to my knowledge, also prefers movies of this genre. From his feedback, it seems that Mark also suffered from the same problem as Scott as he only agreed with less than half of the recommendations. Looking at Mark’s friend list, Scott also appeared as one of the friends but not the best friend.

By incorporating the genre features (that were used in the GA system), this problem would have been avoided. This therefore suggests that although the genre features were shown to have little impact on prediction accuracy in chapter 4, they have an effect on the relevance of recommendations as shown in the cases of Mark and Scott above. Joseph’s comment in table D.5 also echoed this view.

D.1.6. Experiments with Explicit Feedback

In addition to the above feedback, 4 users: Mark, Paddy, Joseph and Julia provided feedback for each movie item recommended to them.

- Mark (user id = 12) stated in terms of ‘agree’ and ‘not agree’
- Paddy (user id = 28) stated in terms of ‘agree’, ‘not agree’ and ‘don’t want to see’
- Joseph (user id = 9) gave the actual rating (between 1 and 5)
- Julia (user id = 19) stated in terms of ‘agree’ ‘so so’ and ‘not agree’

This therefore gave rise to a further experiment to incorporate these users’ explicit feedback into the system in order to observe the performance of the system.
As Joseph was the only user to provide actual ratings for the recommended movies, this posed a problem as to how to convert the feedback from the other 3 users into numerical ratings. The following decisions were made:

- For items that received ‘agree’, use the predicted rating (either 4 or 5) given to the items.
- For items that received ‘not agree’, use a rating of 2. By stating ‘not agree’, the user did not want to give a rating of 4 or 5 for these items. Of the three remaining ratings (1-3), the median, a rating of 2, was used.
- For Paddy’s ‘don’t want to see’, use a rating of 1. It is assumed that by stating ‘don’t want to see’, the user would have given the movies a rating of 1.
- For Julia’s ‘so so’, use a rating of 3 (she feels indifferent about the items).

These additional ratings were incorporated into the training set, increasing the number of training items for Mark from 61 to 105, Paddy from 88 to 162, Joseph from 68 to 83 and Julia from 57 to 142. The experiments were performed using all 37 users with the parameter values outlined in table D.1.

![Figure D.15: Comparison between the original and after feedback – best results with zero tolerance](image-url)
The results showed that with the additional feedback from users in place, the prediction accuracy with zero tolerance for 20 out of 37 users increased, see figure D.15. More importantly, 3 out of the 4 users who provided explicit feedback were amongst those 20. For the other user, Paddy, whose accuracy did not improve after feedback was incorporated, this may suggest that the mapping used to convert his feedback into numerical ratings was not suitable. A mechanism to integrate feedback into the system should be devised which would yield better and more consistent results. With at-most-one tolerance, the prediction accuracy for most users remained the same after feedback was incorporated, see figure D.16. This was expected, as, for example, a shift from a rating of 4 to 5 would have affected the results obtained with zero tolerance but not those obtained with at-most-one tolerance. Note that similar trends were seen in average and worst results.

**D.1.7. Online Recommender System Architecture**

So far, the experiments carried out in this thesis have been performed off-line using pre-collected data. This section now describes a framework for implementing an online recommender system using the ClusterWeight algorithm. The system architecture is shown below in figure D.17.
The server is responsible for managing the database which holds all user profiles. A user profile includes demographic information, movie ratings, the user’s current position together with his/her current best friend and feature weights describing his/her recommendation preferences. In order to make the system scalable, it is proposed that the recommendation engine should take the form of a Java applet, hosted by the server but executing locally on the user’s machine. This would distribute the processing load required by the system, see figure D.17.

For each active user, at the beginning of each session, the server sends a selected set of user profiles (including the profile of the active user’s current best friend) to be used as the current Profile set in the recommendation engine. There are many ways to choose the set and one such way would be to select users whose current position is close to that of the active user and perhaps a few random users to encourage the active user to explore the swarming space, see figure D.18.
Figure D.18: All current positions on the server with possible Profile set for user A encircled.

Once the current profile set has been received by the recommendation engine on the user’s machine, the ClusterWeight algorithm (figure 6.11) is executed and runs continuously until the end of the session. The maximum number of iterations is not set and it is up to the user to specify when he/she would like to receive recommendations. At such point in time, the system uses the current neighbourhood of users to compute predicted ratings on specified movies or to provide a list of recommendations for the user. Note that the user is able to update his/her profile at any time as this is stored locally. The updates can be in the form of feedback on recommended movies, new ratings or modifications to demographic information. At the end of each session, the local profile is synchronised with the server, updating it with the user’s current position, his current best friend and any recommended movies together with profile changes made during the session.

It is anticipated that this architecture will enable the construction of viable future commercial systems, based on the algorithms developed in this thesis. Such a commercial system is considered to be beyond the scope of this research.
Appendix E

Preliminary Work on Fashion and Sizing Recommendations

E.1. Fashion Advisor

Chua's study of shopping behaviours for women's clothes in Singapore (Chua 1992) has suggested a sharp contrast between self-serve mass marketing shops i.e. shops that sell mass production ready-to-wear clothes and exclusive shops that only stock highly priced designer clothes. Not surprisingly the service provided at these two kinds of outlets differs greatly.

In shops pitched at the exclusive end of the market, customers will often receive the full attention of sales assistants who try to tailor their service specifically to the client. For this type of business, the client-base is relatively small and select. The result of this is that the sales assistants develop a very good understanding of their customers' preferences. These sales assistants may be better referred to as fashion experts or personal shoppers.

Because they generally work on a one-to-one basis, they treat every client as a preferred customer and will not only give much more personal and relevant advice but they often work closely with the client and may even "dress" them. What sets them apart is their detailed knowledge of the industry and keen perception of style and fashion. Unfortunately, their service is not cheap and is usually beyond the means of many people.

By contrast, self-service shops such as High Street retail chains do not offer this degree of service. Although they account for a large proportion of the market, customers have to choose their own clothes and queue to pay at the tills. Sales assistants are available to deal with any small issues that may arise, such as stock availability, size enquiries or perhaps product descriptions. The mass-market approach of these businesses focuses on profit, which means that other factors such as quality of service are compromised and consequently sales assistants do not spend large amounts of time solely on one particular customer.
In order to implement a virtual fashion advisor, it is essential that we first define what a fashion advisor is. There are three types of fashion advisors:

- **Sales assistants**: they may be able to make simple suggestions but the large majority of their knowledge is acquired through observation and awareness e.g. which items in the shop are selling particularly well, what customers are wearing, features in magazines etc. Often their suggestions are inferred from their own past experiences with other customers.

- **Fashion experts**: they are the people with great deal of experience and knowledge about fashion. They know what type of clothes would improve customers appearance and in what colours. Because of this, they tend to be open to new ideas and may be more original in their approach to creating a new look. Their broad knowledge allows them to work with more conventional designs as well as unorthodox items.

- **Friends**: our own shopping companions can also be classified as fashion advisors. Although they are the group that has the least experience in giving advice on fashion, familiarity is their key advantage. They can cast an opinion based on more than just your preferences. They know more about your personality and can use that to make suggestions. Also, because the peer relationship is two-way, it may be possible that you will take on new ideas from your friends' personalities and the ways they dress.

**E.2. Sizing Advisor**

The very concept of size as a reliable method of relating a particular garment to someone's body is questioned by Treleaven (2004). The size system for representing how large women's clothes are in the U.K. is taken as an example. The article shows how several women who take the same size clothes actually have significantly varying body shapes. In the U.K., a size 12 might be identified by a combination of bust/waist/hips measurements of 34, 28, and 36 inches respectively. Furthermore, these are only the median points for the range of measurements that may be considered size 12.

The general lack of a rigid and well-defined standard puts the consumer at a disadvantage. People do not know if a particular item of clothing will fit unless they try it on, and
because of this uncertainty it is not unusual for them to take the same item but in 3 different 'sizes' into the fitting room. This is the root of a huge problem for retailers as it has been recorded that a considerably large part of clothes bought are ‘returned’ simply because they did not fit.

There are also "parallel" sizing systems in Europe and the United States for example. To compound the confusion, fashion retailers may interpret these systems as guidelines in whichever way they wish. This creates four different size ranges:

- National Size Ranges - UK, Japan, USA, ...
- Retail Size Ranges - Gap, JCPenny, Benetton, Armani, ...
- Manufacturers Sizes - the actual size of individual garments e.g. shirt, dress, hat, ...
- Customer's Body Size and Shape - customer's actual size

Shop assistants will generally be able to assist with 'sizing', but their scope would normally be limited to the measurement system adopted by their company. This still means that we have to adopt a trial and error approach when it comes to shopping for clothes.

This myriad of sizing standards and scales is only the start. The shape of the body is another major issue to consider when deciding whether an item of clothing is well fitted. Typical female body shapes are tall and slender, petite, full figure, pear-shaped and hourglass shaped. By identifying the body shape, one can then work out how the garment will drape. This in turn will imply the style and cut of clothes that would be suitable for a particular figure.

However, shape alone may not be enough to determine the dimensions of a suit that would fit perfectly. Sometimes other, less quantitative factors might be taken into consideration, such as posture or mannerisms.

The last important factor that also needs to be considered is the ‘ease’ associated with each garment. This means a customer’s exact measurements are still not enough to establish the perfect size for this customer. There are two basic types of ease:
• Wearing ease: the extra material needed for the garment to make the wearer move around comfortably. This, however, depends on customers’ fit preference – how a customer likes to wear a particular item of clothing e.g. close fitted, fitted, semi-fitted, loose fitted and very loose fitted. This will determine the number of inches that will be added to the actual body measurements.

• Design ease: the amount of room or slack given in relation to the garment design. Some designs can have no, or minus ease, such as bathing suits, lingerie.

Because the nature of ease is a personal one, it is very difficult for advisors to give advice on the precise size of the item of clothing without customer having to try it on.

E.3. Real vs. Virtual Advisors

Table E.1 below shows the comparison between real and virtual advisors.

- Possible
- Not possible

<table>
<thead>
<tr>
<th>Tasks</th>
<th>Human Advisors</th>
<th>Virtual Advisors</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Sales Assistants</td>
<td>Experts</td>
</tr>
<tr>
<td>Available 24 hours</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Deep knowledge of fashion</td>
<td>Not always</td>
<td>✗</td>
</tr>
<tr>
<td>Knowledge of personal preference</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Anonymity</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Shopping for friends</td>
<td>✗</td>
<td>Not always</td>
</tr>
<tr>
<td>Universal sizing advice</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Build up personal relationship</td>
<td>✗</td>
<td>Not always</td>
</tr>
</tbody>
</table>

Table E.1: Comparison between real and virtual advisors

- The virtual advisor is available 24 hours a day, 7 days a week. That is one of the key benefits of Internet-based technology - the ability to provide service to users even outside normal working hours. This is something that cannot be expected of
to be available all the time is that there may be users in other time zones, especially those in ones which differ greatly from the local time zone.

- A deep and thorough knowledge of fashion and the industry is not something that can be expected of friends and shop assistants. An expert will have worked in the field for many years and built up a deep knowledge of fashion over that time. The same will apply to an advisor system, as it will employ knowledge discovery techniques on the data it gathers to generate new rules about fashion.

- As we have discussed, if fashion advice is to be of any value, it needs to be tailored for the customer. Due to lack of experience and knowledge, the average sales assistant would not be able to provide much help. However, part of the service that a fashion expert provides involves understanding the client and working closely with him or her to develop an informed opinion and make better recommendations. The advice system conducts similar activities where the user initially registers, and it will continually acquire new knowledge through its interaction with him or her.

- Sometimes the individual may wish to remain anonymous for various reasons. This is not something that can be achieved if the user chooses to ask a sales assistant or a fashion expert for suggestions.

- In the scenario where the user wants to buy an item for someone else, e.g. a gift, then it would be advantageous to know about the fashion preferences the recipient has. This obviously is an area in which a sales assistant can provide the least help.

- The advice system will have an extensive database of measurements relating to various sizing systems adopted by a wide selection of companies and countries. Therefore it will be able to provide information on which size of garments would be most appropriate for the user.

- Fashion advisors will always maintain a level of professionalism whereas the relationship with friends is much closer. The same applies with the virtual advisor. The closeness of the relationship between the advisor and the user is determined by him or her.
The database design for the fashion advisor part of the system is much more complex and has yet to be finalised. There are far more issues that need to be taken into consideration for it. For example, the schema needs to hold user preferences, profiles, lifestyle and consumer habits, including previous selections as well as supporting information such as demographic data. Moreover, the database must store the rules that have been mined or deduced from the data. Another collection of data that will form a large part of the database are those relating to items of clothing that will be held in the "stock" of the system, or product inventory.

E.4. User Interface (Avatar)

Recently various programmers have tried to build human attributes into virtual avatars. But virtual avatars can only move and speak for the time being in spite of the fact that a human being has 5 senses: sight, sound, smell, taste and touch. Of course, in the near future real avatars with the 5 senses will be created. In addition, it is important for avatars to have their own personalities just like Kyoko Date, the world's first virtual avatar, who has her own character and family. At the present time, many self-learning systems exist and we need to incorporate them into virtual avatars. If we can do so, there is a possibility that real avatars will appear who can learn everything themselves.

I supervised Akira Sato on his Masters project - an implementation of a user interface for the virtual fashion advisor. During the course of his project, Akira produced three drawings as initial ideas for the appearance of our fashion advisor. We wanted to gauge people's opinions on these prototypes before going ahead with our implementation so we conducted a survey. Eight colleagues, aged between 23 - 31 years, were asked to choose one woman out of the three that they liked the most. They were also asked to consider the one from whom they would prefer to seek advice on fashion. We observed some very interesting reactions and opinions from the interviewees. Everyone referred to each drawing as a 'person' and referred to them as real people when giving an explanation for their choice. "She looks a bit too young" and "she looks quite intimidating - I would be quite scared to ask her for advice" were a few comments that were received.
General Comments on each girl are shown below in table E.2.

<table>
<thead>
<tr>
<th>Age: 17-19</th>
<th>Nationality: Japanese</th>
<th>Attributes: Young, pretty</th>
<th>Votes: 3 out of 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Good:</td>
<td>Fun and relaxed - fits the role of shopping companion/friend.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bad:</td>
<td>Looks too young. Many people do not feel that they would be able to trust her advice on important occasions, such as business dinner or job interview.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Age: 20-22</th>
<th>Nationality: Japanese</th>
<th>Attributes: sophisticated, pretty</th>
<th>Votes: 1 out of 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Good:</td>
<td>Warm and friendly appearance - makes people feel comfortable shopping with her.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bad:</td>
<td>Again, looks too young.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table E.2: Virtual Fashion advisor survey

It was decided that having a group of fashion advisor characters from which users may choose would be a better approach, than having just a single "perfect" character to cater for all users. Amongst the conclusions gathered from the survey, it was felt that the characters should represent a broad range of people in terms of age, gender and race. Results from our survey also concluded that the hairstyle of the first one presented a fun and relaxed character, and the mouth of the second conveyed friendliness, whilst the eyes of the third presented a more serious and industrious side.