5-2011

Swarm intelligence for clustering dynamic data sets for web usage mining and personalization.

Esin Saka
University of Louisville

SWARM INTELLIGENCE FOR CLUSTERING DYNAMIC DATA SETS FOR WEB USAGE MINING AND PERSONALIZATION

By

Esin Saka
B.S. (CEng), Middle East Technical University, Turkey, 2002
B.S. (Math), Middle East Technical University, Turkey, 2002
M.S., Middle East Technical University, Turkey, 2005

A Dissertation Submitted To the Faculty of the
Graduate School of the University of Louisville
in Fulfillment of the Requirements
for the Degree of

Doctor of Philosophy

Department of Computer Engineering and Computer Science
University of Louisville
Louisville, Kentucky

May 2011

SWARM INTELLIGENCE FOR CLUSTERING DYNAMIC DATA SETS FOR WEB USAGE MINING AND PERSONALIZATION

By

Esin Saka
B.S. (CEng), Middle East Technical University, Turkey, 2002
B.S. (Math), Middle East Technical University, Turkey, 2002
M.S., Middle East Technical University, Turkey, 2005

A Dissertation Approved On

April 27, 2011

by the following Committee

Dissertation Director
Dr. Olfa Nasraoui

Dr. Adel S. Elmaghraby

Dr. James H. Graham

Dr. Mehmed M. Kantardzic

Dr. Joanna L. Wolfe

DEDICATION

Dedicated to my parents

Nuran Saka and Erol Saka

and to my sister

Ernur Saka

ACKNOWLEDGEMENTS

First, I would like to thank my advisor Olfa Nasraoui for believing in my potential and for her endless support. Without the guidance and the freedom that she has provided, this experience would not have been the same.

My journey at the University of Louisville (UofL) started with an email from Tamer Inanc, a professor at UofL, and I would not be here without his introduction. Then I had great internships at Yahoo! and Microsoft Research. Thanks to Michael Rosett and David Kotchan for their collaboration and guidance during my internships. Moreover, thanks to the CRA-W Grad Cohort and Suzan Evans for emphasizing the importance of internships, because internships have changed my pathway.

My experience at the Knowledge Discovery and Web Mining Lab would not have been the same without our discussions, collaboration, seminars, and dinner meetings. Thanks to my colleagues Jaafar Ben-Abdallah, Artur Abdullin, Nurcan Durak, Basheer Hawwash, Gurmit Kaur, Carlos Rojas, Sofiane Sellah, Maha Soliman, Zhiyong Zhang, and Leyla Zhuhadar. I appreciate all my professors who have helped me progress. I would also like to thank the rest of my dissertation committee members, Adel Elmaghraby, James Graham, Mehmed Kantardzic, and Joanna Wolfe for giving me the opportunity to learn from them.

During my journey, several friends have provided sometimes a smile, and at other times a hand to hold on to... Although I am not able to list all the names here, I would like to thank each one of them. It was more fun with Melanie Rudolph. Frank Chen provided a different way to think. Sertan Girgin generously shared his experience and opinions. Maciej Pietrzak showed me how to walk through some rocky fields. Gökçe Yıldırım was on my side whenever I needed her. Burçin Sapaz provided moral support. Finally Sinan Kalkan was there for me every step of the way.

Most of all, I am so grateful to my family, in particular my mother Nuran, my father Erol, and my sister Ernur for supporting me in every possible way unconditionally and selflessly. Sevgili Anneciğim, Babacığım ve Ernurcuğum: Sizi çok seviyorum!

Last but not least, I would like to acknowledge the importance of science and discovery in my life for filling me with curiosity, excitement, and fulfillment...

ABSTRACT

SWARM INTELLIGENCE FOR CLUSTERING DYNAMIC DATA SETS FOR WEB USAGE MINING AND PERSONALIZATION

Esin Saka

April 27, 2010

Swarm Intelligence (SI) techniques were inspired by bee swarms, ant colonies, and most recently, bird flocks. Flock-based Swarm Intelligence (FSI) has several unique features, namely decentralized control, collaborative learning, high exploration ability, and inspiration from "dynamic social" behavior. Thus FSI offers a natural choice for modeling dynamic social data and solving problems in such domains. One particular case of dynamic social data is online/web usage data which is rich in information about user activities, interests and choices.

This natural analogy between SI and social behavior is the main motivation for the topic of investigation in this dissertation, with a focus on Flock based systems which have not been well investigated for this purpose. More specifically, we investigate the use of flock-based SI to solve two related and challenging problems by developing algorithms that form critical building blocks of intelligent personalized websites, namely, (i) providing a better understanding of the online users and their activities or interests, for example using clustering techniques that can discover the groups that are hidden within the data; and (ii) reducing information overload by providing guidance to the users on websites and services, typically by using web personalization techniques, such as recommender systems. Recommender systems aim to recommend items that will be potentially liked by a user.

To support a better understanding of the online user activities, we developed clustering algorithms that address two challenges of mining online usage data: the need for scalability to large data and the need to adapt clustersing to dynamic data sets. To address the scalability challenge, we developed new clustering algorithms using a hybridization of traditional Flock-based clustering with faster K-Means based partitional clustering algorithms. We tested our algorithms

on synthetic data, real UCI Machine Learning repository benchmark data, and a data set consisting of real Web user sessions. Having linear complexity with respect to the number of data records, the resulting algorithms are considerably faster than traditional Flock-based clustering (which has quadratic complexity). Moreover, our experiments demonstrate that scalability was gained without sacrificing quality. To address the challenge of adapting to dynamic data, we developed a dynamic clustering algorithm that can handle the following dynamic properties of online usage data : (1) New data records can be added at any time (example: a new user is added on the site); (2) Existing data records can be removed at any time. For example, an existing user of the site, who no longer subscribes to a service, or who is terminated because of violating policies; (3) New parts of existing records can arrive at any time or old parts of the existing data record can change. The user's record can change as a result of additional activity such as purchasing new products, returning a product, rating new products, or modifying the existing rating of a product. We tested our dynamic clustering algorithm on synthetic dynamic data, and on a data set consisting of real online user ratings for movies. Our algorithm was shown to handle the dynamic nature of data without sacrificing quality compared to a traditional Flock-based clustering algorithm that is re-run from scratch with each change in the data.

To support reducing online information overload, we developed a Flock-based recommender system to predict the interests of users, in particular focusing on collaborative filtering or social recommender systems. Our Flock-based recommender algorithm (FlockRecom) iteratively adjusts the position and speed of dynamic flocks of agents, such that each agent represents a user, on a visualization panel. Then it generates the top-n recommendations for a user based on the ratings of the users that are represented by its neighboring agents. Our recommendation system was tested on a real data set consisting of online user ratings for a set of jokes, and compared to traditional user-based Collaborative Filtering (CF). Our results demonstrated that our recommender system starts performing at the same level of quality as traditional CF, and then, with more iterations for exploration, surpasses CF's recommendation quality, in terms of precision and recall. Another unique advantage of our recommendation system compared to traditional CF is its ability to generate more variety or diversity in the set of recommended items.

Our contributions advance the state of the art in Flock-based SI for clustering and making predictions in dynamic Web usage data, and therefore have an impact on improving the quality of online services.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

In this chapter, we start by introducing the background and motivations behind our research. Then we state our objectives and give a summary of our developed methods and contributions and conclude with an overview of the organization of this document.

1.1 Background and Motivations

Inspiration from nature has driven many creative solutions to challenging real life problems. Many optimization methods have been inspired by such natural phenomena as neural systems, natural evolution, the immune system, and lately particle swarms. In particular, Swarm Intelligence (SI) techniques were inspired by bee swarms, ant colonies, and most recently, bird flocks. Flock-based Swarm Intelligence (FSI) has several unique features, specifically they are decentralized systems, they are collaborative, they are naturally characterized by high exploration, and they are based on "dynamic social" behavior. Thus FSI offers a natural choice for modeling dynamic social data and solving problems in such domains. One particular case of dynamic social data is online/web usage data which is rich in information about user activities, interests and choices. Web usage mining is used for a variety of purposes including mainly the following two tasks: (i) Providing a better understanding of the online users and their activities or interests, for example using clustering techniques that can discover the groups that are hidden within the data; (ii) Reducing information overload by providing guidance to the users on websites and services, typically by using web personalization techniques, such as recommender systems. Recommender systems aim to recommend items that will be potentially liked by a user.

This dissertation explores the suitability of SI for the problem of mining dynamic online social behavior collected in the form of web usage data or in the form of web rating data. Thus in the following sections, we start by introducing the main themes for the components of our research, namely SI-based clustering, dynamic clustering, and web usage mining and personalization.

1.1.1 Swarm Intelligence for Clustering

Clustering is the problem of finding groups in a dataset, according to some data properties and attributes which have a meaning in some context [60, 59]. Since no class information is used to cluster the data, it is called unsupervised learning. In addition to being an interesting and challenging problem, clustering has found success in many applications including customer segmentation in marketing, image segmentation, document organization, web usage mining, personalization, etc.

One of the many different approaches used for clustering is swarm intelligence (SI) and it has been applied to problems that span a variety of domains, such as document clustering and Web session clustering. Swarm intelligence is an artificial intelligence paradigm which is mainly inspired from the dynamics of several societies in nature, such as ant-colonies, bird-flocks, fish-schools, etc. SI is based on the social, collective and structured behavior of decentralized, self-organized agents [65, 146]. Although these agents have a very limited individual capacity, cooperatively they perform many complex tasks. Some distinguishing characteristics of swarm intelligence include: 1) Collaboration: agents in the swarm collaborate or interact with the environment and each other; 2) Collective intelligence: whereas agents in the swarm are mostly unintelligent, the collaborating system, or swarming mechanism results in an intelligent system; 3) Inspiration from nature; and 4) Decentralized control. In this research, we mainly focus on using SI for clustering.

Given the above definition, the most popular swarm intelligence clustering algorithms are:

1. Ant-clustering
2. Particle swarm clustering
3. Flocks of agents-based clustering

There are two main approaches for ant-based clustering. In the first version, data is randomly placed on a grid. Then the ants move around the grid and form clusters by picking up and dropping the data items while moving [81]. Later, this version was improved in [144, 50, 51]. In the second version of the ant clustering algorithm, ANTCLUST, ants represent data items. Initially, none of the ants are assigned to a cluster, i.e. none of the ants have a label. During the clustering process, in each iteration, two randomly selected ants meet each other. Then, according to some defined *behavioral rules*, they may form a new cluster, one of the ants may be assigned to an existing cluster, one of the ants maybe removed from a cluster, or clustering quality measures may be updated [52, 72, 73, 74].

Clustering with particle swarms is based on particle swarm optimization (PSO) [65, 64]. In

the clustering problem, each particle encodes all cluster centroids. In other words, each particle represents a complete clustering solution [6, 142].

More recently, an approach based on flocks of agents, known as FClust, was used for data clustering [129, 128, 117, 116]. The flock-based approach holds a great promise, and being the most recent of all the swarm-based clustering approaches, it has been the least studied. This approach draws its inspiration from bird flocks, such that each agent of the flock represents a data point or record, and each agent is given a position in a 2D or 3D visualization panel. Basically, agents are attracted to similar agents and are repelled by different agents. Moreover, the distance between the agents depends on the similarity between the data items that are mapped to those agents. Therefore, the visualization panel will reflect the similarity relation between the data items. Initially, agents are placed on a planar surface (hereinafter referred to as the *visualization panel*). Then, in each iteration, their speed gets updated according to their neighboring agents, until similar agents start moving together and form clusters. This behavior and the agents' possession of orientation makes FClust particularly useful, not only for clustering, but also for visualization of high-dimensional data.

1.1.2 Dynamic Clustering

One of the unique characteristics of flock-based clustering is its dynamic nature. Agents keep moving on the visualization panel until the algorithm is forced to stop. After a sufficient number of iterations, every state of the visualization panel may provide a clustering alternative for the dataset. This makes FClust suitable for simultaneous clustering and visualization of dynamic datasets.

Unlike conventional clustering, in *dynamic clustering*, data is collected continuously over time, thus the whole data set is not available initially. *Dynamic* domains, such as practically any data generated on the Web, may require frequent costly updates of the clusters (and the visualization), whenever new data records are added to the dataset. The new coming data may be due to new user activity on a website (clickstreams) or a search engine (queries), or new Web pages in the case of document clustering, etc. Additionally, a concept drift in the data records may result in a change of clustering in time [137]. An example of this is the change in the interest of users on an online service. Therefore, clusters may need to be updated, thus leading to the need to mine dynamic clusters. When the data is moreover high-dimensional and sparse, the dynamic behavior makes an already difficult problem even more challenging. Swarm intelligence, mainly ant-based systems, has been used as well as many other methods including repeatedly running a given clustering algorithm, such as K-means,

at each time step [122, 32, 79, 7]. Incremental clustering methods can also be adapted to this task, but they mostly ignore the dynamic nature of the data, which requires distinguishing between old and new data.

1.1.3 Web Usage Mining and Personalization

Every year, several exabytes of digital information are generated [84], which cause *information overload* and make finding information on the Web harder. In particular, online usage data is rich in information about user activities, interests and choices. Modern websites strive to understand the nature of their users in order to improve their services, specifically by helping users find the information that they need or that might interest them. Often, this results in increased profits for the website or online store owner (for instance Amazon, Netflix, etc), in addition to helping serve the user better in seeking their information [example: digital libraries or search engines]. Knowledge Discovery in Data (KDD) techniques have been used increasingly in the last decades in order to mine knowledge from large data sets, and in particular online or web activity data, giving rise to the domain of web usage mining. Web usage mining is used for a variety of purposes including mainly the following two tasks:

- Providing a better understanding of the online users and their activities or interests, for example using clustering techniques that can discover the groups that are hidden within the data.

- Reducing information overload by providing guidance to the users on websites and services, typically by using web personalization techniques, such as recommender systems. Recommender systems aim to recommend items that will be potentially liked by a user.

There are several challenges associated with mining web usage data:

- In the case of web clickstreams, we desire to understand the "mass" user activity patterns without any user-declared profiles, i.e. with only anonymous clickstreams (thus respecting privacy.

- Online usage data tends to have high dimensionality because of the large number (hundreds to thousands) of options that can be clicked, pages that can be viewed, items or products that can be rated or purchased, etc. The high dimensionality makes this data hard to analyze and visualize.

- In addition, online usage data is naturally dynamic because it represents the activities and interests of Human beings that follow dynamic behavior.

Inspiration from the dynamic collaborative behavior of bird flocks can support the design of clustering techniques and recommender systems, i.e. systems that help understand the interests of the users, and systems that can help improve guide their navigation and decision making on the Web.

1.2 Objectives

The natural analogy between Swarm Intelligence systems and social behavior have been the main motivation for the topic of investigation in this dissertation, with a focus on Flock based systems which are a particularly attractive SI systems that possess unique properties for modeling dynamic online social activity, and yet have not been well investigated for this purpose. More specifically, we investigate the use of flock-based swarm intelligence for two related and challenging problems that form critical building blocks of intelligent personalized web servers, specifically

(i) the problem of understanding the online activities of online users by discovering groups or clusters of similar users, and

(ii) the problem of predicting the interests of online users in anticipation of further decision making goals, in particular focusing on collaborative filtering or social recommender systems.

To solve the above two problems, our objectives are to design flock-based solutions to clustering and predicting online user behavior encoded into web usage data. We start by addressing some limitations of the existing Flock-based clustering algorithm FClust, then extend it in two directions to enhance scalability and to enable clustering dynamic data. Then we turn our attention toward developing a social recommendation strategy that is inspired from Flock-based swarm intelligence. The objectives can be summarized as follows.

Objective 1: design a clustering method with improved scalability to handle large data sets.

Objective 2: design a dynamic clustering algorithm that can handle the dynamic nature of online usage data that differs from previous dynamic clustering goals by addressing the following dynamic properties of online usage data: (1) New data records can be added at any time (example: a new user is added on the site); (2) Existing data records can be removed at any time. For example, an existing user of the site, who no longer subscribes to a service, or who is terminated because of violating policies (such as users who post spam or who have been inactive for a long period of time); (3) New parts of existing records can arrive at any time. In other words, the entire data

record associated with a user is rarely available initially. The user's record can change as a result of additional activity such as purchasing new products, returning a product, rating new products, or modifying the existing rating of a product.

Objective 3: design a recommendation system strategy that takes advantage of online user data to recommend interesting items to similar users.

1.3 Summary of Developed Methods

To address some of the limitations of the standard flock-based clustering algorithm (FClust) on Web usage data, we have developed improvements, including the (SPKM+FClust) Hybrid algorithm [8]. The hybrid approach uses a fast K-means based partitional clustering algorithm to initialize the agents' population on the visualization panel, then follows with flock-based clustering. Instead of representing each data point by an agent, each data point is represented by the nearest cluster centroid that was discovered by the partitional clustering algorithim during the initial phase. The result is a much smaller agent population size compared to the entire data set. Our hybrid approach reduces the quadratic complexity of FClust to linear complexity and performs similarly to FClust with fewer iterations for clustering high-dimensional data such as Web usage data. Our experiments confirm the superiority of the proposed hybrid approach, both in terms of quality of the final results and a significantly reduced computational cost.

We developed a dynamic clustering algorithm *Dynamic-FClust* that solves the dynamic clustering problem for dynamic data. The most interesting scenario for using this dynamic clustering approach in real life is when a data record is a set of item ratings by a user, a session of viewed web pages, or a transaction of purchased items. In this case, an attribute of a record may correspond to one rating or to one item. Our approach works by adjusting the agent population in response to each one of the following exhibitions of dynamic behavior in the underlying data set: (1) a new data record that is added; (2) an existing data record that is removed; (3) a new part of an existing data record is received, or an existing part of this old data record is modified. We have tested our algorithm on synthetic dynamic data, and on a data set consisting of real online user ratings for movies. The resulting algorithm is shown to handle the dynamic nature of data without sacrificing quality compared to a traditional Flock-based clustering algorithm that is re-run from scratch with each change in the data.

We used the flocks of agents-based Swarm Intelligence paradigm to develop a new recommender system technique. In this approach, each user from the usage data is mapped to one agent on

the visualization panel. The Flock-based recommender algorithm (FlockRecom) iteratively adjusts the position and speed of dynamic flocks of agents, such that each agent represents a user, on the visualization panel. Then it generates the top-n recommendations for a user based on the ratings of the users that are represented by its neighboring agents. The algorithm works similarly to FClust. However, there is no explicit clustering process or goal. Instead, in each iteration, recommendations for a user are generated or updated using similar users that are represented by its neighboring agents on the visualization panel. In this approach, the agents move continuously through the visualization panel unless they are forced to stop. This dynamic behavior of the agents allow a continuous exploration of different recommendation options, which means that FlockRecom can provide dynamic recommendations, and the additional exploration allows it to include more diversity among its recommended items. Variety or diversity is important in an environment where the users make repeated visit, such as social networks and online music and video channels. In such an environment, without continuous or dynamic exploration, the same recommendations would be repeatedly suggested because the system would eventually converge and stagnate on one set of choices. Our initial experimental results show that FlockRecom is a promising approach for recommendation in dynamic environments, thus having potential applications in social networking platforms [130].

Our recommendation system was tested on a real data set consisting of online user ratings for a set of jokes, and compared to traditional user-based Collaborative Filtering (CF). Our results demonstrated that our recommender system starts performing at the same level of quality as traditional CF, and then, with more iterations for exploration, surpasses CF's recommendation quality, in terms of precision and recall. Another unique advantage of our recommendation system compared to traditional CF is its ability to generate more variety or diversity in the set of recommended items.

1.4 Contributions of this Dissertation

Our main contributions can be summarized as follows:

- New (K-means+FClust) and (SPKM+FClust) hybrid algorithms were developed. Our hybrid approach reduces the quadratic complexity of FClust to linear complexity, and performs similarly to FClust, but has the advantage of fewer iterations for clustering large, high-dimensional data such as web usage data. Hybrid algorithms were tested on several datasets including UCI machine learning data sets and Web server logs and our experiments confirmed their superiority, both in terms of quality of the final results and computational costs [128, 129].

- A dynamic clustering and visualization approach that can perform dynamic clustering was developed and tested. This approach can handle the arrival of not only one data record at a time, but also one attribute at a time, in any order for one data record. It can also handle the modification or updating of an individual attribute (such as one item's rating) from a record and even the removal of a data record.

- A new recommender system approach called the flocks-of-agents based recommender system (FlockRecom) was developed. The results were compared to the traditional user-based nearest neighbor Collaborative Filtering and FlockRecom was more successful at providing variety in the recommendations without losing recommendation quality [130].

- Under special simplifications we formulate the flocking behavior as a Gradient Descent optimization that minimizes a criterion that agrees with the observed behavior in flocking in the case of attraction, namely grouping (agents getting closer to each other) and alignment. In addition it is minimizing the error between ideal and agent distance values, therefore seeking a visualization of the original data onto the 2D panel that is as faithful as possible. The mathematical derivations are included under the Future Visions in the Conclusions Chapter. According to our knowledge, this a mathematical interpretation of flocking behavior that has not been accomplished so far. And we plan to pursue it further in the future.

1.5 Organization of this Document

This dissertation is organized in six chapters. Chapter 2 presents the literature review, starting with a review of algorithms for clustering, incremental clustering, data visualization, swarm intelligence based clustering, and flocks of agents based clustering, then continuing with multi-agent systems and intelligent agents on the Web. Chapter 3 presents our contributions on FClust, such as the improved FClust, FClust Annealing, (K-means+FClust) hybrid algorithm, (SPKM+FClust) hybrid algorithm. Chapter 4 presents the flocks of agents based recommender system and experimental results. In Chapter 5 we present a dynamic data clustering algorithm using the flocks-of-agents based approach with experiments on synthetic and real-life datasets. Finally, our conclusion, in Chapter 6, summarizes the study and discusses the vision for the future.

CHAPTER 2

LITERATURE REVIEW

In this chapter, we present the background information related to our research. We start by introducing clustering, dynamic clustering, and similarity measures which can be used for clustering. Then we continue with explaining clustering using swarm intelligence, and provide a deeper survey on flocks of agents-based clustering. At the end of the chapter, intelligent agents, and multi-agent systems are shortly overviewed, concluding with agents on the web and recommendation systems.

2.1 Clustering

Clustering is the problem of finding groups in a dataset, according to some data properties and attributes which have a meaning in some context [60]. Since no class information is used to cluster the data, it is also called unsupervised learning. In addition to being an interesting and challenging problem, clustering has many applications including:

- **Text Clustering:** One of the applications of text clustering is the Web search engines. The documents on the Web are clustered for different purposes such as fast document retrieval, more related advertisements, better recommendations, etc. [35, 40]. Text clustering is also used for document organization, mostly in digital library applications.

- **Customer Segmentation in Marketing:** Finding similar customers may be used for designing targeted products, offering personalized services, etc. Additionally, insurance companies may use customer clusters for risk analysis. Recommendations by Amazon[1] is an example.

- **Web Usage Mining:** Clustering users on the Web provides a better ground for mining knowledge from the user activity logs and personalized services. As a result, web pages can be re-designed and personalized, information retrieval systems (i.e. search engines) may be improved, etc. [95, 103].

[1] http://www.amazon.com/

- **Image Segmentation and Image Clustering:** Image segmentation has a wide area of applications from health sciences to video storage. Image segmentation can be used for sickness decision support in health sciences. Additionally, image clustering may be used for video and/or image compression and content-based image retrieval [151].
- **Outlier Detection:** With the help of clustering, certain anomalies may be determined in a dataset[105].

There are different ways to classify clustering algorithms [60, 136, 62]. The classical classification distinguishes hierarchical approaches from partitional approaches. However, more recent classifications also include density-based clustering [22]. In *hierarchical clustering*, a nested series of clusters are organized as a hierarchical tree, which is called a *dendogram* while *partitional clustering* methods produce only one level of clustering and divide the dataset into non-overlapping subsets (i.e. clusters) such that each data item belongs to one cluster. Although some density based clustering algorithms are partitional, the *density based* clustering algorithms directly seek dense locations in the data set, without explicitly seeking to divide the data into subsets [22, 97]. Another aspect distinguishing hierarchical approaches is the clustering process: If the algorithm assigns each data item in one cluster and merges clusters until a stopping criteria is met, it is called an *agglomerative* approach, whereas in the *divisive* approach, initially all data items are assigned to a single cluster and clusters are split until a stopping criterion is met. A third aspect relates to using the attributes sequentially or simultaneously. If all the attributes are used at the same time during clustering, it is called *polythetic*. If the attributes are used sequentially one at a time for clustering, it is called *monothetic*. More aspects and a survey about clustering can be found in [60] and [22].

Many different techniques has been used to solve the clustering problem such as K-Means, artificial neural networks (ANNs), self-organizing maps (SOM), etc. [60]. Lately, swarm intelligence has also been applied for clustering [129, 6, 52].

2.1.1 Similarity Measures

One of the factors deeply affecting the success of clustering is the measure used for similarity. Therefore, the choice of similarity measure is important. Similarity can be defined based on the distance (the closer the items are the more similar they are). Two popular distance metrics are the Manhattan (L1 or city-block) and Euclidean distances.

Given that each data record x_i in the A dimensional real number space R^A, $i = 1, ..., n$, is represented by a vector $x_i = x_i^1, ..., x_i^A$, the Manhattan (L1 or city-block) distance between two data records x_i and $x_j \in R^A$ is given by

$$Manhattan\_distance(x_i, x_j) = \sum_{k=1}^{A} |x_i^k - x_j^k|, \tag{1}$$

The Euclidean distance is given by:

$$Euclidean\_distance(x_i, x_j) = (\sum_{k=1}^{A} |x_i^k - x_j^k|^2)^{1/2} = ||x_i - x_j||_2, \tag{2}$$

The Minkowski distance is a generalization of both the Euclidean and Manhattan distances [49]:

$$Minkowski\_distance(x_i, x_j) = (\sum_{k=1}^{A} |x_i^k - x_j^k|^p)^{1/p} = ||x_i - x_j||_p, \tag{3}$$

where p is a positive integer. It represents the Manhattan distance when $p = 1$ and Euclidean distance when $p = 2$.

The Manhattan based (L1) similarity of two data records x_i and x_j is given by

$$sim(x_i, x_j) = 1 - \frac{1}{A} \sum_{k=1}^{A} |x_i^k - x_j^k|, \tag{4}$$

where A denotes the number of attributes and x_i^k denotes the k^{th} attribute of data record x_i. When the data is linearly normalized to $[0, 1]$, the Manhattan based similarity is the same as the 1-norm similarity, which is used in (4).

The cosine and Jaccard similarities are common in Web mining. The Jaccard coefficient measures similarities between two sets A and B using the ratio of the number of elements contained in the intersection and the union of the sets [49]:

$$Jaccard\_coefficient(A, B) = \frac{|A \cap B|}{|A \cup B|} \tag{5}$$

The cosine measure is commonly used to represent similarity between two high-dimensional data vectors, such as in the text mining and Web mining domains. Let v_i and v_j be two A dimensional data vectors. Their cosine similarity is defined as [49]:

$$Cosine\_similarity(v_i, v_j) = \frac{v_i \cdot v_j}{||v_i||||v_j||} \tag{6}$$

where the inner product $v_i \cdot v_j$ is the standard vector dot product and $||v_i||$ is the norm defined as:

$$v_i \cdot v_j = \sum_{k=1}^{A} v_i^k v_j^k \tag{7}$$

$$||v_i|| = \sqrt{v_i \cdot v_i} \tag{8}$$

All the above similarities consider each attribute to be different from and unrelated to every other attribute. This is not realistic, since in some problems, attributes tend to be semantically related. One example is web pages or URLs. For this reason, in Web mining, a similarity measure called syntactic similarity or Web session similarity used the Web site structure to first define similarities between URLs [104].

Let the i_{th} user session be encoded as an A-dimensional binary attribute vector v_i with the following property:

$$v_i = \begin{cases} 1, & \text{if user i accessed URL j} \\ 0, & \text{otherwise} \end{cases} \tag{9}$$

The entire Web site is modeled as a tree, where the nodes represent different URLs. The tree structure is similar to a file directory structure, with an edge connecting one node to another if the URL corresponding to the latter is hierarchically located under that of the former. Thus, a URL u_i is represented as:

$$u_i = root/un_1/.../un_z \tag{10}$$

where un_i represents a node on the tree structure and z is a finite integer.

Given the representation of a Url u_i, p_{u_i} denotes the path traversed from the root node, the main page, to the node corresponding to the i_{th} URL. $|p_i|$ indicates the length of this path.

The *syntactic* similarity between the ith and jth URLs is defined as

$$S_u(i,j) = min(1, \frac{|p_i \cap p_j|}{max(1, max(|p_i|, |p_j|) - 1)}) \tag{11}$$

This similarity measures the amount of overlap between the paths of the URLs. The similarity values lies in $[0, 1]$. The overall web session similarity is based on (11) and is given by:

$$web\_session\_similarity(v_i, v_j) = \frac{\sum_{k_i=1}^{A} \sum_{k_j=1}^{A} v_i^{k_i} v_j^{k_j} S_u(k_i, k_j)}{\sum_{k_i=1}^{A} v_i^{k_i} \sum_{k_j=1}^{A} v_j^{k_j}} \quad (12)$$

A comparison between these different similarity measures is given in Table 1. On the table, attributes are classified into two: 1) Quantitative: Attributes which have numerical values, such as 1, 0.001, etc. 2) Qualitative: Qualitative attributes are the ones such as color being blue, red, etc. Web session logs can be considered as both qualitative and quantitative attributed datasets. If a session is represented as a set/bag of words, we can consider it qualitative. However, if a session is represented as a vector, as given in Equation 9, then we say that the qualitative dataset is represented in a quantitative way. Among all the similarity measures given on Table 1, Manhattan is the simplest and has the minimum computational cost. Although Euclidean is more costly than Manhattan, it exaggerates the distance, creates bigger distinction between similar and dissimilar items. Minkowski distance-based similarities are mostly preferred with low dimensional datasets with quantitative attributes. Although qualitative attributes can be represented with numbers, since this does not present an order or nominality, Euclidean and Manhattan distance based similarities are not very suitable for qualitative data. When the data items are (i.e. can be) represented as sets, The Jaccard coefficient, is the ratio of the intersection of two data items to the unification of these data items. Therefore, it is suitable for qualitative datasets. One way to represent qualitative attributes is by representing existence in a data record, as given in Equation (9) or representing frequency instead of only existence. Thus, cosine similarity is suitable for datasets with both qualitative and quantitative attributes. Compared to cosine similarity, Web session similarity can perform a more detailed similarity analysis, since it compares how similar even different attribute values are. However, Web session similarity is computationally more costly than cosine.

In our experiments, we used Manhattan distance-based similarity for experiments with numerical data, because it was suitable to these datasets and we were able to compare our work with previous studies, since previous studies used Manhattan distance-based similarity. However, for the Web session datasets, which is a high-dimensional sparse dataset, Minkowski distance-based similarities returned close-to-zero similarities. Therefore, they were not preferred. We chose the cosine similarity because it was proved to yield sufficient conditions for the centroid update equations to converge with SPKM[40], whereas the Web session based similarity or Jaccard coefficient were not proved to work with SPKM. In the future, we plan to investigate different similarity measures

wherever needed.

In the case of ratings datasets, the similarity can be computed via comparing the rankings, as in the case of Pearson's correlation coefficient.

Given two users u and v, Pearson's correlation coefficient is defined in (13) [78].

$$Pearson\_correlation(u,v) = \frac{\sum_{i \in C}(r_{u,i} - \overline{r}_u)(r_{v,i} - \overline{r}_v)}{\sqrt{\sum_{i \in C}(r_{u,i} - \overline{r}_u)^2}\sqrt{\sum_{i \in C}(r_{v,i} - \overline{r}_v)^2}} \tag{13}$$

where C is the set of items that are rated by both of the users u and v; u_i and v_i are the rating of item i by user u and v respectively; and $\overline{r}_u$ and $\overline{r}_v$ are the average of ratings of u and v respectively.

In the algorithms, if the correlation is negative, we consider it as 0 (zero). A disadvantage of the Pearson correlation is, given a subset user u_s of a user u, such that u_s is almost the same as u except that it does not include all the ratings, the similarity of the u_s is not very close to 1. In other words, Pearson defavors/disapproves new users with small number of ratings. These new users will not be similar to any established users enough.

TABLE 1. Comparison of different similarity measures.

Similarity Measure	Quantitative Data	Qualitative Data	High Dimensional Sparse Data	Advantage	Disadvantage
Manhattan	Yes	No	No	Simple and cost efficient	Common 0 similarity on high dimensional data
Euclidean	Yes	No	No	Effective with low dimensional data	Common 0 similarity on high dimensional data
Jaccard	No	Yes	Yes	Effective with qualitative data	Not suitable for continuous valued attributes
Cosine	Yes	Yes	Yes	Effective with high dimensional sparse data	High computational cost
Web Session	NA	NA	Yes	Specialized for datasets with semantically related attributes (e.g. URLs)	Higher computational cost than cosine

2.1.2 K-Means Algorithm

The K-Means Algorithm is a popular, simple, unsupervised, learning algorithm for clustering. [85]. In this iterative, partitional clustering approach, each cluster is associated with a cluster centroid and each data point is assigned to the cluster with the closest centroid.

The K-Means Algorithm, listed in Algorithm 1, aims to minimize an objective function, consisting of the sum of squared errors or distances between the data records and K cluster centroids, as given in Equation (14), by iteratively updating the cluster centroids and assigning data to the most similar centroid, until the total error between the data records and the assigned cluster centroids converges. Note that this procedure forms clusters with n-dimensional hyper-spherical boundaries, where cluster centroids represent the centers.

$$E = \sum_{j=1}^{d} \sum_{i=1}^{n} ||\chi_i^j - c_{\pi(\chi_i)}^j|| \tag{14}$$

In 14 n is the number of data records, d is the number of attributes and $\pi(\chi_i) = cluster\ to\ which\ \chi_i\ is\ assigned.$

The complexity is $O(n * K * I * d)$ where n is the number of points, K is the number of clusters, I is the number of iterations, and d is the number of attributes.

Algorithm 1 K-Means Algorithm

Input: Dataset $\chi \in \Re^d$ where $|\chi|=n$; number of clusters: $K \leq \sqrt{n}$.
Output: A partition of the dataset into K disjoint clusters $\gamma_1, ..., \gamma_K$.

1: Read data records χ_i, $i = 1, ..., n$.
2: Arbitrarily select K records as centroids out of the n data records.
3: **repeat**
4: **for all** Data record χ_i **do**
5: Find the closest centroid c_i using Euclidean Distance.
6: Assign data record χ_i to the cluster γ_i.
7: **end for**
8: **for all** Cluster γ_j **do**
9: Update its centroid $c_j = \frac{\sum_{\chi_i \in \gamma_i} \chi_i}{||\sum_{\chi_i \in \gamma_i} \chi_i||}$
10: **end for**
11: **until** stopping criterion is met.

2.1.3 The Spherical K-Means Algorithm

The Spherical K-Means Algorithm (SPKM) is a popular algorithm for clustering high dimensional and sparse data, such as transactions, text, and web usage or clickstream data [40]. SPKM is particularly suitable for data sets where the cosine similarity is more appropriate than the Euclidean distance (used in the K-Means), due mainly to the asymmetric nature of the features/dimensions. Asymmetric features are such that, when comparing two data vectors, the *presence* of the feature (positive value) is considered to be much more important than its absence (zero value). The SPKM, listed in Algorithm 2, aims to maximize the average cosine similarity between the data vectors and K cluster centroids, by iteratively updating the cluster centroids and assigning data to the most similar centroid, until the total cosine similarity between data vectors and assigned cluster centroids converges, as given in Equation 15.

$$L = \sum_{\chi_i} \chi_i^T c_{\pi(\chi_i)} \tag{15}$$

where $\pi(\chi_i) = argmax_k \chi_i^T c_k$.

Algorithm 2 Spherical K-Means Algorithm

Input: Dataset $\chi \in \Re^d$ where $|\chi|=n$; number of clusters, $K \leq \sqrt{n}$.
Output: A partition of the dataset into K disjoint clusters $\gamma_1, ..., \gamma_K$.

1: Read sessions.
2: Normalize data records to be of unit length.
3: Arbitrarily select K records as centroids out of the n data vectors.
4: **repeat**
5: **for all** Data vector χ_i **do**
6: Find the most similar centroid c_i using cosine similarity.
7: Assign data vector χ_i to the cluster γ_i.
8: **end for**
9: **for all** Cluster γ_j **do**
10: Update its centroid $c_j = \frac{\sum_{\chi_i \in \gamma_i} \chi_i}{||\sum_{\chi_i \in \gamma_i} \chi_i||}$
11: **end for**
12: **until** stopping criterion is met.

2.1.4 Cluster Validation

Cohesion-Separation: Inter- and Intra- Cluster Similarities

One of the methods to evaluate and compare the quality of clustering results is measuring the cluster *cohesion (compactness, tightness)*, which determines how closely related the data items in a cluster are, and cluster *separation (isolations)*, which determines how distinct or well separated a cluster from other clusters [136]. Thus, clustering results may be evaluated by computing the average *inter-cluster* and *intra-cluster* similarities. The *inter-cluster* similarity is the similarity between clusters and determines the cluster separation (isolation), whereas the *intra-cluster* similarity is the similarity between data points within a cluster and determines the cluster cohesion. A higher intra-cluster similarity and lower inter-cluster similarity represent better clustering quality.

Silhouette Coefficient

The method of silhouette coefficient combines both cohesion and separation. It is given in the means of distance in [136], and if we modify it for similarity, the silhouette coefficient is computed for a data item i as follows:

- For the item i, compute the average similarity to all other data items in its cluster. Call this value a_i.
- For the item i and any other cluster not containing i, calculate the item's average similarity to all the items in the given cluster. Find the maximum such value with respect to all clusters and call this value b_i.
- For the i, the silhouette coefficient $s(i)$ is computed as given in Equation (16).

$$s_i = \frac{a_i - b_i}{\max(a_i, b_i)} \tag{16}$$

The value of silhouette coefficient may vary between -1 and 1. The most desired case is silhouette coefficient being 1. Positive values as close to 1 is better because $\lim_{a_i->1, b_i->0} s_i = 1$. A negative value shows that i is more similar to the items in another cluster than the items in its own cluster.

Davies-Bouldin (DB) Index

Davies-Bouldin (DB) index is a measure which represents the average similarity between each cluster and its most similar one, in a clustering result [48]. Let C_i and C_j be two clusters in a clustering result. Let s_i be a measure of dispersion of cluster C_i, and d_{ij} be a dissimilarity (distance) measure between clusters C_i and C_j. Then, a similarity measure R_{ij} between clusters C_i and C_j is defined to satisfy:

- $R_{ij} \geq 0$.
- $R_{ij} = R_{ji}$.
- If $s_i = 0$ and $s_j = 0$ then $R_{ij} = 0$.
- If $s_j > s_k$ and $d_{ij} = d_{ik}$ then $R_{ij} > R_{ik}$.
- If $s_j = s_k$ and $d_{ij} < d_{ik}$ then $R_{ij} > R_{ik}$.

A possible choice of R_{ij} is given in Equation (17) [48].

$$R_{ij} = \frac{s_i + sj}{d_{ij}} \tag{17}$$

Then the DB index is defined as given in Equation (18) [48]

$$DB_{n_c} = \frac{1}{n_c} \sum_{i=1}^{n_c} R_i \tag{18}$$

where $R_i = \max_{i=1,...,n_c, i \neq j} R_{ij}, i = 1, ..., n_c$.

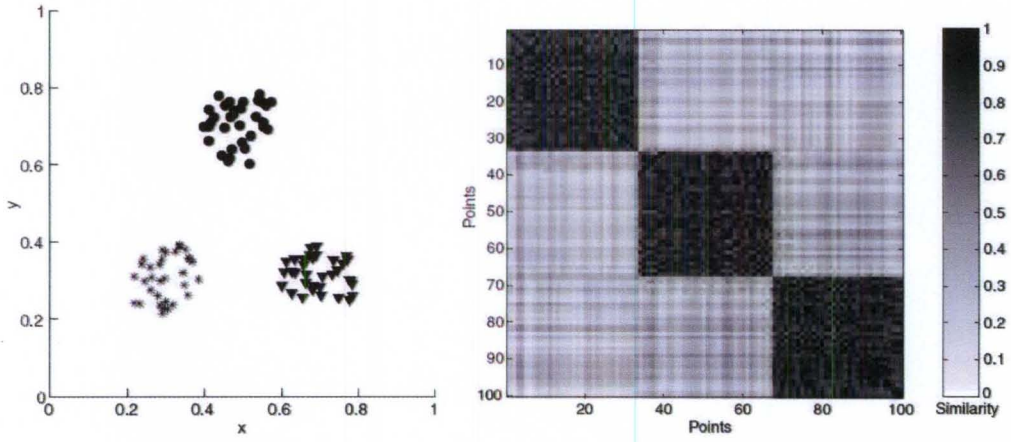
Since DB index represents the average similarity between each cluster and its most similar one and it is desirable for the clusters to have the minimum possible similarity to each other; a clusterings that minimizes DB is more desirable.

Similarity Matrix Approach

Given the similarity matrix and cluster labels, another cluster evaluation approach is to compare the similarity matrix to an ideal similarity matrix [136]. In an *ideal similarity matrix*, the similarity is 1 if items are in the same cluster and 0 otherwise. Therefore, if we sort the rows and columns of the similarity matrix (and the ideal similarity matrix) so that all items belonging to the same cluster are together, then the ideal similarity matrix has a block diagonal structure.

High correlation between the ideal similarity matrix and the similarity matrix is desired, because it indicates that similar items in the same cluster and clusters are well separated.

There is also a visual way of judging clustering by its similarity matrix [136]. As explained above order the similarity matrix with respect to clusters, and then plot it. If the color gets darker as the similarity approaches to 1, in the ideal case, blocks would be observed around the diagonal, representing clusters and the rest would be light colored. Clouds on the plot represent correlated clusters, which is not desired. Figure 1(b) shows a plotted similarity matrix for well-separated clusters given in Figure 1(a) and extracted by K-means [136].



(a) Well-separated clusters.

(b) Similarity matrix sorted by K-means cluster labels and colored.

Figure 1. Visual validation using sorted and plotted similarity matrix [136].

Precision and Recall, MSE, Robust Cardinality

As in the case of most applications, in Web usage mining, clusters can be modeled via prototypes. In the input to Web usage mining, a set of URLs visited by a specific user at a limited time is called a session. Therefore, in this case, a prototype may be called a *profile* and will be a set of URLs, which may optionally include the frequencies of URLs.

The metrics precision and recall are defined as follows [126, 12]: *Precision* is the proportion of retrieved items that are really relevant and *recall/coverage* is the proportion of all items, known to be relevant, that are retrieved. The discovered profiles can also be considered as frequent itemsets, or patterns, and provide one way to form a summary of the input data. As a summary, profiles represent a reduced form of the data that is at the same time, as close as possible to the original input data. This description is reminiscent of an information retrieval scenario, in the sense that profiles

that are retrieved should be as close as possible to the original session data. Closeness should take into account both (i) precision (a summary profile's items are all correct or included in the original input data, i.e. they include only the true data items) and (ii) coverage/recall (a summary profile's items are complete compared to the data that is summarized, i.e. they include all the data items) [104].

To assign a session to a cluster a distance measure must be defined, typically to be inversely related to similarity, e.g. $distance = 1 - similarity$:

Two different measures can be used to evaluate the final prototypes [97]:

- The mean squared error (MSE) or average dissimilarity between cluster prototypes and data records.
- Robust cardinality.

The mean squared error or average dissimilarity, for the i^{th} cluster is given by:

$$\sigma_i^{*2} = \frac{\sum_{s^{(k)} \in \chi_i} d_{ik}^2}{|\chi_i|} \tag{19}$$

where $s^{(k)}$ is the k^{th} session, χ_i is the set of sessions assigned to the i^{th} cluster, and d_{ik} is the distance from $s^{(k)}$ to χ_i.

The robust cardinality for the i^{th} cluster is given by:

$$N_i^* = \sum_{s^{(k)} \in \chi_i} w_{ik} \tag{20}$$

where w_{ik} is the *robust weight*, given by:

$$w_{ik} = exp(-\frac{d_{ik}^2}{2\sigma_i^{*2}}) \tag{21}$$

where σ_i^{*2} is a scale measure that assesses the dispersion of the sessions around the cluster prototype. w_{ik}, which is in $[0, 1]$, is high for inliners/good data and low for outliers/noise.

Cluster vs. Class Labels

When there are class labels available, a cluster error may be computed by comparing the cluster and class labels. The procedure given in Algorithm 3 checks whether each item pair has the same class labels if they have same cluster labels or not.

Additionally, surveys on cluster validation were presented in [47, 48, 136].

Algorithm 3 Cluster Error Computation Algorithm

Input: Dataset with class labels and cluster labels from FClust output.
Output: Cluster error -a real number between 0 and 1-.

1: $Error \leftarrow 0$
2: **for all** Data record pairs (i, j), where $i \neq j$ **do**
3: **if** i and j have the same class label, but different cluster labels **then**
4: $Error++$
5: **else if** i and j have different class labels, but the same cluster label **then**
6: $Error++$
7: **end if**
8: **end for**
9: Return $Error/Number\ of\ pairs$

2.1.5 Dynamic Clustering

Given the clustering definition in Section 2.1, we define *dynamic clustering* as follows:

Dynamic clustering is the process of extracting clusters in a dataset in which the new data items becomes available, some new clusters may emerge, some existing clusters may disappear, or some data items may change in time. To the best of our knowledge, there have been no studies on dynamic clustering with "dynamic items", i.e. data items changing in time.

However, there are some related studies. One of them is known as *incremental clustering* or *online clustering* in which, data becomes available in batches over time [43, 7, 32, 79, 77]. One common approach for incremental clustering is repeatedly applying a classical clustering algorithm, in which case a high computational cost may be prohibitive. Additionally such an approach does not differentiate new data from existing data. Incremental clustering was initially proposed as a solution to clustering on very large databases [43] and unlike dynamic clustering, does not consider change in "existing" data items. Moreover, in some application, the whole dataset may be available initially. However, if the dataset is huge, batches are formed by sampling, and clustering is performed incrementally [43]. In IncrementalDBSCAN, DBSCAN is used for incremental clustering and compared to DBSCAN [43]. IncrementalDBSCAN can handle emerging and disappearing clusters, but does not handle data items that change in time. One common approach for incremental clustering is to repeatedly apply a classical clustering algorithm every time more data arrives or change in data occurs, in which case a high computational cost may be prohibitive. Additionally such an approach does not differentiate new data from existing data. In another algorithm called leader-follower clustering (LFC), a K-means type clustering is used [42]. Basically, when a new item is available, the closest cluster centroid is determined. If the new item is similar enough to this cluster centroid, the item is assigned to the cluster represented by that centroid. Otherwise, a new cluster is generated. The LFC algorithm is given in Algorithm 4 [80]. In another approach, clusters are generated according to the initially available data batch and as the new data arrives, new data is assigned to existing clusters [32].

Ant colony-based swarm intelligence algorithms are some of the most common approaches for solving incremental clustering in SI. In a study which uses ant clustering to cluster continuous data, despite the lack of a benchmark set, the results were promising [122]. Similarly, another study divided a dataset into different groups, assumed that each group was available at a given time step and used ant clustering to update the clusters [79]. Also in [32], the authors ran the fuzzy K-means algorithm to observe initial clusters and used ants, which were moving in a three dimensional space,

Algorithm 4 Leader Follower Algorithm [42]

```
1: repeat
2:    Read new item i
3:    Find the closest cluster center c
4:    if Distance is above threshold then
5:       Create new cluster
6:    else
7:       Add i to cluster represented by c
8:       Update c
9:    end if
10: until No more data is available
```

for clustering the new coming data.

We will define the difference between incremental clustering and dynamic clustering as the capability to handle generating new clusters as needed and updating the cluster for an item which changes in time. Therefore, it can be said that dynamic clustering includes incremental clustering as a special case. However, dynamic clustering is more suitable than incremental clustering for some dynamic environments, such as the Web. On the Web, some clusters may emerge while some others may disappear. Additionally, attributes of a data item may change in time, as in the case of an update of a Web page or a change in the likes and dislikes of a user on the Web. Therefore, clustering user profiles is an interesting domain for dynamic clustering.

Another approach that may be similar to dynamic data clustering is stream clustering. Some application areas of stream clustering can be listed as network intrusion detection, weather monitoring, emergency response systems, stock trading, electronic business, telecommunication, planetary remote sensing, and web site analysis [31]. More information on stream clustering can be found in [31, 96] The difference between stream clustering and dynamic clustering is: in stream clustering each new coming input is a *new* data item, where as in dynamic clustering, new input can be an update of an existing data item. For example, assume there is a set of users and new inputs are user-url pairs, user-query pairs, user-item pairs in short. The user may be a new user *or an existing user.* Since stream clustering algorithms does not store the data, they cannot handle "dynamic data".

As a new clustering paradigm, dynamic data clustering is compared to other clustering paradigms in more detail in Section 5.2.

Limitations of the Current State of the Art in Dynamic Data Clustering

Dynamic clustering is an interesting subject to work on and requires more improvement. Current limitations include:

- To the best of our knowledge, there is no clustering research considering a data item changing in time.
- An efficient and low cost dynamic clustering algorithm for clustering high dimensional sparse data is not available.
- Better algorithms are required which can handle emerging and disappearing clusters.
- There is not enough benchmarks to compare different algorithms.
- More study is needed to define dynamic, incremental, online, and stream clustering.
- There is a lack of an approach which can cluster and visualize data simultaneously.
- There is no research using flocks of agents for dynamic clustering.

2.2 Data Visualization

Recent improvements in the hardware and software have allowed storing more and more information each day. Every year, exabytes of digital information is generated [84], which have made data mining and visualization even more valuable. *Data visualization* is the study of the visual representation of data [45, 118]. It is accepted that data visualization includes scientific visualization (i.e. use of computer modeling and simulation in scientific and engineering practice) and information visualization (i.e. visualization of abstract and heterogeneous datasets such as as large-scale collections of non-numerical information) [118]. One of the advantages (or aims) of visual data exploration is to include the Human in the data mining process; by presenting the data in some visual form. Human perceptual abilities can be used to allow humans to get an insight into the data and interact with the data [63]. Including humans in the data mining process is especially useful when little is known about the data and the exploration goals are vague [63].

Depending on the data type, different visualization techniques can be used. For visualization of datasets with less than 4 dimension (i.e. 1D, 2D, and 3D), some common techniques for data visualization include x-y plots, x-y-z plots, line plots, and histograms [63]. When the data becomes multi-dimensional or unstructured, visualization becomes more challenging. In the case of text or hypertext datasets, data is typically transformed into description vectors [53, 63], whereas for multi-dimensional data, multidimensional scaling can be used via computing item-to-item similarities and assigning each data item a new location in a low-dimensional space, which is suitable for making graphs or plots [25, 34]. Self-organizing maps (SOM) are also a popular way for data visualization and some of its applications include the visualization of web-based social networks [119, 143]. Lately, swarm intelligence has also been used for data visualization [128, 129, 117].

Visualization Evaluation

Publications on visualization show that, visualization does not only have an engineering approach but also has an artistic angle [88, 26, 139, 138, 141]. Therefore, while some people have been evaluating the visualizations from an artistic view and trying to explain why some visualizations are better and more efficient than others, some other researchers have been trying to define visualization metrics. In fact the latter claim that visualization metrics are still an active and challenging research area which needs improvement [88, 26].

Prof. Tufte looks into visualization not only from a scientific view but also from an artistic

angle. In [139, 138]. In [138], he shows the effect of selecting correct visualization parameters and techniques by comparing success and failure via visual representation, where he compares the success in the cholera epidemic in London versus the failure in the decision to launch the Space Shuttle Challenger.

"Clutter and confusion are failures of design, not attributes of information", says Edward Tufte in [140], (page 51), and presents the fundamental principles of excellent graphics, i.e. analysis and display of data, in [141]:

- **Comparisons:** "Show comparisons, contrasts, differences."
- **Causality, Mechanism, Structure, Explanation:** "Comparisons may lead to reasoning."
- **Multivariate Analysis:** "Show more than one or two variables (i.e. attributes, such as time, location, direction, temperature and date simultaneously)."
- **Integration of Evidence:** "Completely integrate words, numbers, images and diagrams."
- **Documentation:** "Thoroughly describe the evidence. Provide a detailed title, indicate the authors and sponsors, document the data sources, show complete measurement scales, point out relevant issues."
- **Content counts most of all:** "Analytical presentations ultimately stand or fall depending on the quality, relevance, and integrity of their content."

In [26], Richard Brath proposed metrics for 3D visualizations:

- **Number of Data Points:** The number of data items represented on a visualization screen at an instant.
- **Data Density:** The number of data points per pixel. In Equation 22, the number of pixels does not include the pixels in the window borders, menus, etc.
- **Number of Simultaneous Dimensions and Cognitive Overhead:** As the number of dimensions increase, the cognitive complexity increases. Tufte also mentions that unnecessary information should not be visualized.
- **Dimensional Score and Cognitive Complexity:** Dimensional score aims to determine the effectiveness of mapping between data dimensions and visualized dimensions. The higher the

TABLE 2

Dimensional Scoring Summary [26].

Mapping	Score
$n - to - 1$	$3Xn$
$1 - to - 1$ general	2
$1 - to - 1$ intuitive	1
preexisting representation	0

dimensional score, the higher the cognitive complexity. The computation of score is summarized in Table 2. In the table, n is the dimension. General mapping is a mapping where the user needs to remember what the dimension on visualization represent, on the contrary, an intuitive mapping is one where the user is not expected to remember, such as the attribute size is named size on the visualization screen. The last row is for preexisting common visualizations such as market share and histograms. However, this score does not handle $n - to - m$ mappings, where $1 < m < n$.

- **Percentage of Occlusion:** Occlusion percentage is the ratio of the number of data points completely obscured over the number of data points. The less the occlusion is the better.

- **Reference Context and Percentage of Identifiable Points:** This measures the capability to show location of data items in reference to the other data items and computed as given in Equation 2.2. The higher the ratio is the better.

$$\text{Data Density} = \frac{\text{number of data points}}{\text{number of pixels on the visualization panel}} \tag{22}$$

$$\text{Percentage of identifiable points} = \frac{\text{no of visible data points identifiable in relation to every other visible data point}}{\text{no visible data points}}$$

It should be noted that these metrics are defined for visualization of data itself. Whereas, what we visualize is the relation between data items, the similarity between data items and between data clusters.

Another common method to compare different visual systems is by arranging usability evaluations [151]. In a usability evaluation, a number of people are asked to compare different systems. In our research, dynamic visualization aims to support tracking changes in clusters, such as emerging, merging, separating, and disappearing clusters as well as visualizing the similarity between data

items and between data clusters. Thus, a usability test would compare dynamic FClust to original FClust. A usability test template may be given as follows: Let's assume that we have 5 different synthetic datasets each with the following 5 cases:

- One emerging cluster.
- Two merging clusters.
- One cluster being separated into two.
- One cluster disappearing.
- No change, (i.e. although new data is added in time, there is no change in the cluster number and cluster profiles).

We may select 5 users, with no clustering background. The visualizations of the above 5 cases may be shown to each user either by original FClust or dynamic-FClust. Thus, each user will see 1) Each dataset once 2) 2 or 3 results by FClust 3) 2 or 3 results by dynamic-FClust. The users may be asked to tell:

1. What happened and how easy it was to determined what happened, on a 5-point scale.
2. How easy was it to determine similar groups, on a 5-point scale.
3. How easy was it to compare similarity between different data items, on a 5-point scale.

However, the task should be presented in a better task description. A task description may be given as follows: "For your social sciences class project, you are expected to determine the user profile changes in one month at the University of Louisville web domain. Please determine how are the different user groups are changing in 5 different months. Please also record how many different profiles exist for each month. For each month, mark on the 5-point scale satisfaction sheet 1) Can you tell how similar the user groups are? 2) Can you compare similarity between two random data points. 3) How easy was it to determine what kind of changes happened."

To sum up, some visualization measures that may be used to evaluate dynamic clustering visualization are:

- Number of data points and data density. Number of data items represented per pixel or per unit area.
- Number of dimensions and cognitive overhead.

- Percentage of occlusion.
- Reference context and percentage of identifiable points.
- Usability : User studies are organized where users are asked to evaluate and compare different visualization panels.

Limitations of the Current State of the Art in Data Visualization

Below, we list some of the reasons why more efforts are needed in data visualization:

- Better algorithms are needed which are capable of clustering and visualizing *simultaneously.*
- There is a lack of algorithms on simultaneous dynamic clustering and dynamic data visualization.

2.3 Simultaneous Data Visualization and Clustering

One of the advantages of FClust over many other clustering algorithms is the capability of visualizing and clustering data simultaneously. Thus, another criterion of the expected Dynamic-FClust algorithm is the ability to visualize the similarity between data items and clusters. In [88], Self-Organizing Maps (SOM) was used to project text documents onto a Tree fractal. Figure 2 shows a sample visualization by the system [88]. This system focuses on visualization and not clustering, and the provided visualization was static. Alternative ways for cluster visualization in self-organizing maps are presented in [87], however, the experiments were on a very small dataset, with 16 items and 13 attributes and includes no dynamic clustering or visualization. Later, the SOM algorithm was scaled up to be able to deal with clustering and visualization of large amounts of high-dimensional data [67]. Figure 3 shows a sample visualization by the system. However, this algorithm did not propose dynamic clustering and visualization, either. In [21], the neurons on the perimeter of the map are kept active and a dynamic update of SOM is proposed to achieve incremental clustering. However, the inner nodes are static, and after training SOM, an ascendant hierarchical clustering was applied to cluster the neurons. Thus, similar clusters from different learning periods may be separated on the map, and incremental SOM may not visualize the data as original SOM does. Additionally, some other points that needed more clarification and were not addressed in [21] are: 1)What was the size of the map (i.e. the initial number of neurons) initially for SOM and incremental SOM, 2)Was the size of the map updated for each learning period for the classical SOM, 3)If not, why not? 4) How does the incremental SOM handle splitting and merging clusters. Thus, better explanation and further experiments are needed to show the workings of the proposed approach. Although there have been a few studies on dynamic SOM [82], to the best of our knowledge, no dynamic SOM clustering and visualization algorithms have been published.

Table 3 gives a comparison of different approaches. To sum up, many categorization systems have the inability to perform classification and visualization on a continuous basis or to have new data-items self-organize into the existing clusters (or into new clusters if necessary), unless a new processing and analysis happens. This disadvantage is also present in more recent approaches using Self-Organizing Maps, as in Kohonen maps [122]. Therefore, while a benchmark comparison of the above cited methods should be interesting to explore, any serious comparison will be complicated by the ability of the dynamic FClust algorithm to perform continuous mappings with simultaneous continuous visualization and the inability of the SOM to accomplish it.

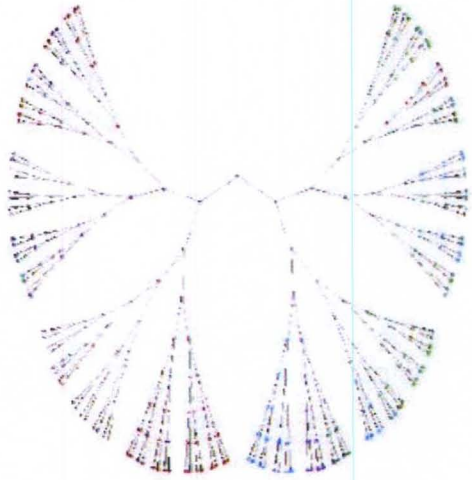


Figure 2. Fractal projection of cancer document vectors [88].

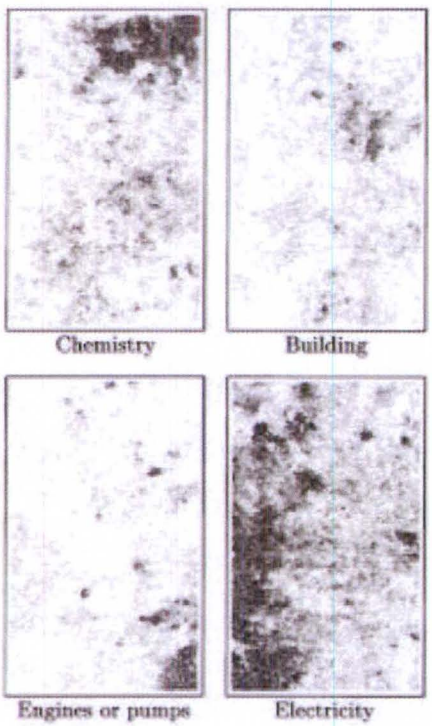


Figure 3. Distribution of four sample subsections of the patent classification system on the document map. The gray level indicates the logarithm of the number of patents in each node. Generated by SOM [67].

TABLE 3. Comparison of different clustering and visualization algorithms.

Algorithm	Incremental Clustering	Emerging/ Disappearing Clusters	Changing Data Items	Cluster No Extraction	Visualization	Dynamic Visualization	Typically Compared to
IncrementalDBSCAN [43]	Yes	Yes	No	Yes	No	No	DBSCAN
Incremental Ant Clustering [122]	Yes	Yes	No	No	No	No	Static Ant Clustering
LFC [42]	Yes	Only Emerging	No	Yes	No	No	K-Means
SOM [88, 87, 67]	No	No	No	Yes	Yes	No	NA
Incremental SOM [21]	Yes	Only Emerging	No	Yes	No	No	SOM
FClust[117]	No	No	No	Yes	Yes	No	K-Means
Dynamic FClust	Yes	Yes	Yes	Yes	Yes	Yes	FClust

2.4 Swarm Intelligence-based Clustering

"The whole is greater than the sum of its parts" - Aristotle

Swarm intelligence is an artificial intelligence paradigm based on social, collective and structured behavior of decentralized, self-organized agents [65, 146]. Algorithms in this domain mainly depend on an inspiration from nature, in particular from social insects like bees, ants, termites; flocks of birds, and fish schools. These animals have a very limited individual capacity. Yet, cooperatively, they perform many complex tasks such as searching for and storing food and flying collectively over long distances. The characteristics of swarm intelligence are:

- *Collaboration :* Agents in the swarm collaborate or interact with the environment and with each other.
- *Collective intelligence:* Whereas agents in the swarm are mostly unintelligent, the collaborating system, or swarming mechanism results in an intelligent system.
- *Inspiration from nature:* Agents' properties tend to be derived from analogous creatures such as ants, bees, or birds.
- *Decentralized control:* Agents behave and interact without a centralization mechanism.

The five basic principles of swarm intelligence are [89]:

- **Proximity principle:** The group should be able to perform simple space and time computations.
- **Quality principle:** The group should be able to respond to quality factors in the environment.
- **Principle of diverse response:** The group should not allocate all of its resources along excessively narrow channels.
- **Principle of stability:** The group should not change its behavior with every change in the environment.
- **Principle of adaptability:** The group should change its behavior if it is beneficial.

In this chapter, we will mainly focus on using SI for clustering. Most common SI algorithms for clustering are:

1. Ant-clustering

2. Particle swarm clustering

3. Flocks of agents clustering

In the following subsections, general information about ant-clustering and particle swarm clustering is provided. Following this section, we will discuss clustering using flocks of agents in detail.

2.4.1 Particle Swarm Clustering

Clustering with particle swarms is based on particle swarm optimization (PSO) [65, 64], which was initially inspired from bird flocks. Though the initial motive was modeling human social behavior, PSO later became a very popular search and optimization technique.

In PSO, the population is a group of particles and each particle is a candidate solution to the problem. Therefore in the swarm intelligence concept with PSO, a swarm is a solution *set.* Each particle flies through a multi-dimensional problem space, and every position represents a different solution. After each move, the position is evaluated by a fitness function as shown in Line 8 of Algorithm 5. The fitness function aims to evaluate the performance of each particle which is the closeness of the solution represented by the particle to the global optimum solution. The personal best solution which is the best position visited by the particle so far is calculated and kept (Line 10 of the Algorithm 5). The particle's best position and the global best, which is the best solution found in the neighborhood, are used for computing the particle's new location (Line 12 of the Algorithm 5). Depending on the definition of neighborhood, there exist two different versions of PSO: 1)*gbest:* the neighborhood is the entire swarm 2) *lbest:* a swarm is divided into overlapping neighborhoods of particles and the best particle is determined in the neighborhood.

The PSO Clustering algorithm was first applied in image clustering [108, 109], which took the number of clusters as input and used the gbest version of PSO. In the clustering problem, each particle is constructed from all cluster centroids. In other words, each particle represents a clustering solution [6, 142].

A hybrid model combining PSO clustering with the K-means clustering algorithm was presented in [142], where one of the particles was initialized with the result of K-means. Another (PSO+K-means) hybrid model was used for document clustering in [35, 36], where the results illustrated that the hybrid PSO algorithm can generate more compact results than K-means and PSO. A survey and a modified PSO-based clustering algorithm was presented in [6].

Algorithm 5 The PSO Clustering Algorithm

Input: Dataset and the number of clusters K.
Output: Clustered data.

```
1: Initialize each particle to contain K random cluster centroids.
2: for iteration=1 to max do
3:   for all particle i do
4:     for all data record x do
5:       Calculate the Euclidean distance of x to all cluster centroids in i.
6:       Assign x to the closest cluster.
7:     end for
8:     Calculate the fitness of the particle.
9:   end for
10:  Find the global best position (among all particles) and personal best position of each particle.
11:  Update the velocity of each particle based on the global and personal best positions.
12:  Update the cluster centroids based on the particles' velocities.
13: end for
```

2.4.2 Ant Clustering

There are mainly two approaches for ant-based clustering. In the first version, *Ant Clustering Algorithm (ACA)*, data is randomly placed in the environment, which is generally a two-dimensional plane with a square grid. As shown in Algorithm 6, the ants move around the grid, via a random walk or jumping to form clusters by picking up, transporting, and dropping the data items while moving around [81]. The picking and dropping operations are influenced by the similarity of the surrounding data items and the density of the ant's local neighborhood $f(x_i)$. Generally, the size of the neighborhood is 3×3. The probability of picking up increases when the data item is surrounded by dissimilar data items or the density of the neighborhood is low. Similarly, the probability of dropping increases when similar data items are encountered. After ACA is stopped, a post-processing algorithm such as an agglomerative clustering algorithm is run for cluster retrieval [50]. More recently, performance analysis and strategies for increased robustness were presented [50, 51]. Improvements were also proposed by adding a progressive vision scheme and including pheromone on the grid cells [144]. A good review can be found in [52].

In the second version of the ant clustering algorithm, ANTCLUST, each ant represents a data item. Initially, none of the ants are assigned to a cluster, i.e. none of the ants have a label. Then, during the clustering process, in each iteration, two randomly selected ants meet each other. According to some defined *behavioral rules*, they may form a new cluster, one of the ants may be assigned to an existing cluster, one of the ants may be removed from a cluster, or clustering quality measures may be updated [72, 73, 74]. The basic idea is that agents who carry similar data items

Algorithm 6 The Ant Clustering Algorithm (ACA)

Input: Dataset.
Output: Clustered data.

```
1: Randomly scatter data items on the grid.
2: Randomly scatter ants on the grid.
3: for iteration=1 to max do
4:   for all ant a_i do
5:     if a_i is unladen and a_i's grid position is occupied by item x_i then
6:       Calculate f(x_i) and calculate prob_{pick-up}(x_i) using f(x_i).
7:       if prob_{pick-up}(x_i) ≥ random() then
8:         Let a_i pick up item x_i.
9:       end if
10:    else if a_i is carrying item x_i and a_i's grid position is empty then
11:      Calculate f(x_i) and calculate prob_{drop}(x_i) using f(x_i).
12:      if prob_{drop}(x_i) ≥ random() then
13:        Let a_i drop item x_i to i's current grid position.
14:      end if
15:    end if
16:    Move a_i to a randomly selected, neighboring, unoccupied grid position.
17:  end for
18:  iteration++
19: end for
```

attract each other, while agents who carry dissimilar agents repel each other, which results in the formation of groups. A general outline of the ANTCLUST Algorithm is given in Algorithm 7 and behavioral rules are given below.

Ant Behavioral Rules in ANTCLUST:

1. **New nest creation:** If two ants without clusters meet each other and they are similar enough, they form a new cluster.

2. **Adding an unlabeled ant to an existing nest:** If a labeled ant a_i meets an unlabeled ant a_j, and if they are similar enough, a_j is labeled with the same label of a_i, i.e. added to the cluster of a_i.

3. **Positive meeting between two nest-mates:** If two ants a_i and a_j have the same labels and they are similar enough, then the quality measures are increased for these ants and the cluster they belong to.

4. **Negative meeting between two nest-mates:** If two ants a_i and a_j have the same labels and they are not similar enough, then the quality measures are decreased for these ants, and the ant with smaller quality measure is unlabeled.

5. **Meeting between two ants of different nests:** If two ants a_i and a_j have different labels

but they are similar enough, then the quality measures are decreased for these ants and the cluster they belong to. Then, the ant belonging to the smaller nest is moved to the other ant's bigger nest.

6. **Default rule:** If none of the above applies, do nothing.

Algorithm 7 The Ant Clustering Algorithm ANTCLUST

Input: Dataset.
Output: Clustered data.

1: Map ants to data items and initialize ants' quality measures. Initially ants do not have any labels, i.e. they do not belong to any cluster.
2: **for** iteration=1 to max **do**
3: Randomly choose two ants and apply the behavioral rules above.
4: iteration++
5: **end for**
6: Delete nests that do not contain enough ants.
7: Reassign ants without labels to the most similar nests.

Another clustering algorithm called AntTree uses the ability of building mechanical structures of ants and builds a tree structure [11]. In this version, each data to be clustered represents a node of the tree and the algorithm searches for the optimal edges.

2.5 Using Flocks of Agents for Data Visualization and Clustering

A recent swarm intelligence approach used a *flock of agents*. One of the definitions given for a *flock* is "*a number of animals of one kind, esp. sheep, goats, or birds, that keep or feed together or are herded together*"[2].

"The motion of a flock of birds is one of nature's delights" according to Craig Reynolds who has simulated this phenomenon in computer animation, where the bird-like, birdoid object was called *boid* [124]. One of the biggest differences between a particle and a boid in simulation is that boids have *orientation*, which makes them suitable for *data visualization* as well as clustering.

Studies about flocks of agents in computer science have mainly started with simulating moving bird flocks, based on two balanced and opposing behaviors of natural flocks, namely, 1) Desire to stay close to the flock, and 2) Desire to avoid collisions. These are simulated in the following three behaviors [124].

Natural Bird Flock Behaviors:

1. **Collision Avoidance/Separation:** Steering away from the other boids to avoid collision.
2. **Alignment/Velocity Matching:** Aiming to match the moving direction (i.e. heading) and speed to that of nearby flockmates.
3. **Cohesion/Flock Centering:** Attempting to adjust steering toward the average position of local flockmates and to stay close to the neighbors.

While cohesion and velocity matching represent the attraction forces, which keep the boids together, collision avoidance formed the rejection/repelling force. Other studies also tried to present behavioral rules and model collective behavior of animals [55, 33]. Later studies also focused on visualizing data using flocks of agents. Each individual boid represented one data item and a fourth behavior was added to represent moving with similar data items [120]:

4. **Information Flocking:** Attempting to move with similar boids.

The fourth behavior is pretty similar to the second behavior, velocity matching. However, in the fourth behavior, the aim is not moving together with all neighbors, but only with the ones similar enough to form a group. This behavior provided a suitable ground for using flocks of agents for data visualization and offered a motivation for data clustering.

[2] http://dictionary.reference.com/browse/flock

It should be noted that, just as a flock can be formed of birds, it can also be formed by other boids such as fish, sheep, etc. Therefore, for the sake of generality, in this study, instead of the word boid, we use the word "agent".

The Multiple Species Flocking clustering model (MSF) [37] used flock clustering for data clustering and implemented a distributed multi-agent system. In that study, MSF was compared to Ant clustering and K-Means clustering algorithms. MSF converged faster than ant clustering. However, the clustering results for MSF were manually generated. The user looked at the visualization panel and selected the clusters. The results showed that MSF performed better than K-Means. Lately, in [117], a detailed flock clustering algorithm (FClust) was presented with a stopping criterion and automated cluster extraction algorithm. An application of this approach to Web usage mining can be found in [128]. In the following sections, we will describe the FClust algorithm, discuss its limitations, and suggest improvements with real life application examples.

2.5.1 Flocks of Agents Based Data Visualization

I don't paint things. I only paint the difference between things.

Henri Matisse

Data visualization using flocks of agents is suitable for any kind of data set where one can define a similarity measure between data items. A flock consists of several agents, with each agent representing one data record. As mentioned in Section 2.5, flocks are different from ordinary particles because they have *orientation.* Agents in a flock are attracted to similar agents and are repelled by the different agents. Moreover, the distance between the agents depends on the similarity between the data items that are mapped to those agents. Therefore, the visualization panel visualizes the similarity relation between the data items. Normally, data sets with at most three attributes can be visualized by a simple plot. However, when there are more than three attributes, this becomes harder. In particular, when there is a huge number of attributes, as in web usage data, data visualization becomes a challenging job.

When flocks of agents are compared to other swarm intelligence algorithms, such as ants and particle swarms, we find that flocks are more suitable for data visualization. In the case of ants, data items are moved on a rigidly structured *grid* by ants and placed on the same stack with similar items. However, the distance between ants or data items does not necessarily reflect the similarity between the original data items, as in the case of flocks. The distance between two neighboring agents in a flock is inversely proportional to the similarity between their corresponding data items,

whereas the similarity between two ants only increases the chance of data items being neighbors, but does not define how close/far they are. The distance between two more similar data items may be bigger than the distance between two other items which are less similar. In the case of particle swarms, each particle does not represent one data item, but rather represents a clustering solution itself. Therefore, particle swarms may not be as suitable for data visualization, either.

It has been mentioned that neural networks can also be used for data visualization. However, most neural networks have a rather static structure whereas a flock of agents is inherently dynamic [120]. Also, neural networks are a centralized learning mechanism, whereas agent flocks are decentralized.

2.5.2 Flocks of Agents-Based Clustering

In the clustering with flocks of agents approach [117], each agent represents one data item. Initially, agents are placed on the *visualization panel*, which is a 2 or 3-dimensional continuous space, where x, y (and if applicable z) coordinate values range between 0 and 1. Agents may be placed randomly or some background information can be used to place them. Then, they start moving around. As they meet other agents in a defined neighborhood, they try to remain at an ideal distance to each other, which is determined according to the similarity of the original data items that agents are representing. The more the data items are similar, the smaller the ideal distance will be. Ideal distances are computed for each agent pair once at the beginning of the algorithm based on the intrinsic properties or attributes of the data items. If neighboring agents are further apart than the ideal distance, there will be an attraction force between them and the agents will try to move closer to each other. In contrast, if the distance is less than the ideal distance, then there will be a rejection force, and agents will move apart from each other. Given this basic idea, Algorithm 8 gives the procedure for *Flocks of Agents Clustering (FClust)*.

In steps 1 and 2, the initialization is performed. The velocity vector $\overline{v}$, is a unit vector, (i.e. $||\overline{v}|| = 1$), representing the direction. In step 3, the ideal distances between agents are computed via Equation (23). Later, for each agent i, the neighboring agents that are close enough to i on the visualization panel, are extracted in Line 6, where *d(i,j)* is the 2D Euclidean distance between agents i and j. Then, for each neighbor:

- If the distance between the agents i and j is equal to the ideal distance between them (Line 7), there is no attempt to change i's velocity due to j (Line 8).

Algorithm 8 FClust Algorithm [117]

Input: Dataset.

Output: Visualization of interaction between the data items. Agents corresponding to more similar items are located closer in the 2D visualization panel.

1: Initially place the agents on the visualization panel
2: Initialize velocities of all agents
3: Compute the ideal distances, d_{ideal}, between agents.
4: **repeat**
5: **for** each agent i **do**
6: **for all** j such that $d(j,i) \leq d_{th}$ and $i \neq j$ **do**
7: **if** $d(i,j) = d_{ideal}(i,j)$ **then**
8: $\beta(i,j) \leftarrow 0$
9: **else if** $d(i,j) > d_{ideal}(i,j)$ **then** // attraction
10: $\beta(i,j) \leftarrow 4 \times \left(\frac{d(i,j)-d_{ideal}(i,j)}{d_{th}-d_{ideal}(i,j)}\right)^2$
11: **else** // repulsion
12: $\beta(i,j) \leftarrow -4 \times \left(1 - \frac{d(i,j))}{d_{ideal}(i,j)}\right)^2$
13: **end if**
14: $\overline{v}_{resulting}(i,j) \leftarrow \overline{v}(j) + \beta(i,j) \times \overline{v}_{cap}(i,j)$
15: **end for**
16: **if** $\exists\ j$ such that $d(j,i) \leq d_{th}$ and $i \neq j$ **then**
17: $\overline{w}(i) = \text{normalize}\ (\sum_{j|d(j,i)\leq d_{th}\&i\neq j} \overline{v}_{resulting}(i,j))$
18: **if** The angle between $\overline{v}(i)$ and $\overline{w}(i)$ is less than or equal to 90 degrees **then**
19: $\overline{v}_{next}(i) \leftarrow \overline{w}(i)$
20: **else**
21: $\overline{v}_{next}(i) \leftarrow \overline{v}(i)$
22: **end if**
23: **else**
24: $\overline{v}_{next}(i) \leftarrow \overline{v}(i)$
25: **end if**
26: $amp_{next}(\text{i}) \leftarrow amp_{def} + \frac{d_{th}}{20\times(neighbor\_no(i)+1)}$
27: **end for**
28: **for** each agent i **do**
29: compute new position $p_{next}(i) \leftarrow p_{current}(i) + amp_{next}(i) \times \overline{v}_{next}(i)$
30: **end for**
31: Move all agents to the updated positions and update current velocities.
32: **until** Clusters are formed

- If the distance between the agents i and j is greater than the ideal distance between them (Line 9), an attraction force will move i closer to j, with a more similar velocity to j (Line 10).
- If the distance between the agents i and j is smaller than the ideal distance between them (Line 11), a repelling force will move i further from j, with a less similar velocity to j (Line 12).

In line 14, the velocity effect on i due to neighbor j is computed where $\overline{v}_{cap}(i,j)$ is the unit vector pointing from i to j. Next is the computation of the updated velocity of agent i, $\overline{v}_{next}(i)$,

between lines 16 and 25. First, if i has neighbors, then their resulting velocities on i are summed up and normalized. If the total, normalized velocity $\overline{w}$, does not change the agent's current direction more than 90 degrees, then the updated velocity is assigned as $\overline{w}$. Otherwise the velocity is kept unchanged for the next iteration. Similarly, if agent i does not have any neighbors -note that an agent is not considered to be a neighbor of itself- then the velocity will be kept the same for the next iteration. In line 26, the amplitude is computed depending on the number of neighbors and distance threshold, where amp_{def} is the default minimum amplitude. If the amplitude is too low, it may increase the number of iterations to converge. However, if the amplitude is too high, the agents may move further than the desired location. The minimum amplitude is empirically set to $\frac{1}{5} \times d_{th}$ [117]. At the end of each iteration, the updated agent coordinates are computed, and all the agents are moved to their updated positions simultaneously (lines 28 to 31). Moving agents around the visualization panel is performed until a stopping criteria is met and/or clusters are formed.

Setting the Parameters for FClust

The selection of parameters has a big impact on the convergence of the FClust algorithm. The first parameter is d_{th}, the distance threshold, which defines the neighborhood size (see line 6 in Algorithm 8), and the latter affects the ideal distance via Equation (23) (see line 26 in Algorithm 8) and amplitude. When d_{th} is too small, the agents cannot affect each other, and when it is too high, the algorithm does not converge. One method to compute the ideal distance between two agents, d_{ideal}, is given in Equation (23). If the ideal distances are overestimated, then the clusters cannot be observed on the visualization panel.

$$d_{ideal}(i,j) = \frac{1 - sim(i,j)}{1 - sim_{th}} \times d_{th}. \tag{23}$$

The similarity threshold, sim_{th}, in (23) is computed via Equation (24). If the similarity threshold is too large, then the algorithm will fail to converge, and if it is too small, then different clusters risk being combined into one cluster.

$$sim_{th} = \frac{sim_{average} + sim_{max}}{2} \tag{24}$$

Stopping Criteria for FClust

The most common method for stopping the algorithm is using human experts [120, 37, 117]. An expert keeps watching the visualization panel until stable clusters are formed. At that time, the algorithm is stopped.

However, an automated method was also presented in [117], which used the spatial entropy of the agents relative to their location on a grid imposed on the agent space. In this approach, the visualization panel was mapped onto a $20X20$ matrix. For each cell (i,j) of this matrix, the proportion of $p(i,j)$ of agents that are located in this cell was computed. Then the spatial entropy at iteration t was computed by using Equation (25). If the observed minimum of entropy has not been improved since the last $3 \times n$ iterations, where n is the number of agents, then the algorithm stops. The problem with this criterion is that the entropy may remain unchanged for a long while whenever new neighbors do not meet, but after a meeting occurs, changes may start re-occurring. Thus, the above stopping criterion cannot capture these delayed dynamics, and the algorithm risks to be stopped before convergence. Additionally, how many cells are needed to map the visualization screen and how many iterations are needed to observe minimum entropy change were determined

experimentally and may not work for some datasets.

$$\text{Spatial Entropy}(t) = -\sum_{i=1}^{20}\sum_{j=1}^{20} p(i,j) \times ln(p(i,j)) \tag{25}$$

Cluster Formation in FClust

Algorithm 9 Cluster Formation Algorithm

Input: Agents' coordinates at a stable/converged state.
Output: Clusters of agents.

```
1: for each agent i do
2:   if i's cluster is not assigned then
3:     Form a new cluster c
4:     Assign i to c
5:     for all agent j such that there exists an agent k such that k ∈ c and distance(k,j)≤ d_th and
       sim(k,j) > sim_th do
6:       Assign j to c
7:     end for
8:   end if
9: end for
```

When FClust is run, flocks of agents are visually observable. However, the clusters are not explicitly formed and the data is not yet assigned to clusters. Similar to the stopping criterion, one method of forming clusters is using human experts. The person marks the clusters and assigns agents to the clusters. Since there is a one-to-one mapping between agents and data records, the data will also end up being clustered. In addition to this, an automated procedure was presented in [117], which is given in Algorithm 9. Basically, a new cluster is created for an unlabeled agent. Then the neighboring agents of this cluster are explored, and all the agents which are similar to at least one of the agents in the cluster are inserted into this cluster. New agents are inserted to the cluster until no more agents can be inserted. Then the procedure restarts by creating another new cluster, and stops when all the agents are labeled.

After cluster formation, a post processing phase is needed to cluster the original (input) data, to validate the results, and if possible to interpret the clusters.

Complexity Analysis of FClust

The FClust algorithm, given in Algorithm 8, needs to compare every agent to every other agent in order to compute the ideal distance initially, and then to update the agent's velocity based on its neighboring agents. Despite some complexity reduction suggestions, such as using a neighborhood matrix, [117], the worst case time complexity remains $O(n^2)$, where n is the number

of data records. Similarly, the memory complexity is also $O(n^2)$ to keep the ideal distances, in addition to $O(n)$ memory needed for keeping agent locations, velocities and amplitudes.

Limitations of FClust

Although the experimental results given in [117] were acceptable, we observed that the standard deviations of the number of clusters, the cluster error, and the number of required iterations were high. Moreover, FClust was not successful for each data set. Another disadvantage was the high computational cost which makes FClust unsuitable for many real time applications.

In addition to the above limitations, we observed that convergence strongly depends on the similarity threshold. When the similarity threshold is too high, the algorithm may not converge, and when the threshold is too low, the algorithm may not differentiate between different clusters, and thus may end up combining some of them. Therefore, Equation (24) is not suitable for every dataset. Furthermore, if the data similarity values follow a power low distribution, then the similarity threshold given in Equation (24) will produce a very high similarity threshold. Therefore agents will not be able to form clusters. In other words, the clustering algorithm will not converge. Another problem occurs when there are connecting agents between clusters, meaning the presence of a bridge of data points connecting two clusters. As a result, the clusters are labeled as the same, even though they should be labeled differently. To solve this problem to some extent, an alternative formulation will be presented in Section 3.1. Moreover, the ideal distance formula in [117] requires mostly unique data records. Otherwise, if many similarity values are 1, the ideal distance computation results in an infinite value in Equation 23 because $sim_{th} = 1$. An alternative formulation, which can handle many 1-similarities is given in Section 3.1.1.

2.6 Intelligent Agents

An *agent* is anything that perceives its environment through its sensors and gives a response through its effectors [127], [145]. A human, a robot, a personalized agent on the Web are examples of agents. Figure 4 gives the prototype of an agent.

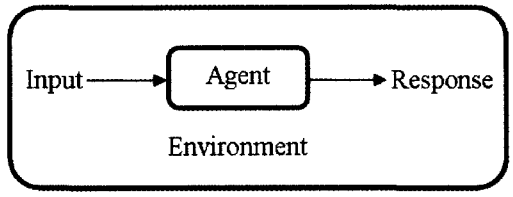


Figure 4. An agent in its environment. Takes input from the outside world and produces an output.

In the computing community, an *agent* is a computer system in its environment, sensing and acting *autonomously* in order to meet its design objectives, in order to meet its goal [145, 147]. That said, the properties of an agent can be defined below [147].

- *Autonomy* is being able to act, i.e. operate, without direct intervention of humans or others and having some control over an agent's internal state and actions [145, 147]. In [93], autonomy also includes being able to adapt to changes in the environment.
- *Reactivity* is the ability of perceiving one's environment and responding in a timely fashion to the changes that occur in the environment in order to manage one's goal [145, 147].
- *Pro-activity* is the ability to display goal-directed behaviour by taking the initiative [145, 147].
- *Social Ability* is the ability of interacting, i.e. communicating, with other agents via some kind of agent communication language in order to meet a goal [145, 147].

It should be noted that, there may not be general recognition of terms like agent, autonomy, intelligent agent, etc. One example is available above: According to some definitions, autonomy also includes reactivity, which may sound awkward.

Every agent is considered to be autonomous according to [145, 147]. An agent is considered to be *intelligent* if it is also reactive, pro-active and social.

2.7 Multi-Agent Systems and Societies of Agents

A multi-agent system is a system in which more than one agent exist. This is a way of distributed computing. Agents may communicate, and agent communication and interaction protocols are some of the design parameters of an agent society.

Flocks of agents, ant colonies are some examples of multi-agent systems. The survey in [110] points to the striking similarities between the Web and natural environments, and gives an overview of applications of artificial life principles to searching and managing techniques, including an agent based approach called *InfoSpider*.

2.7.1 Multi-Agent Learning

Typically, the ability of an agent to learn is a necessary and satisfactory condition for the agent to be described as *intelligent* [145]. There are two types of learning in multi-agent systems [145]:

- *Centralized learning*: which is also called *isolated learning*. Learning is executed by only one agent without any interaction with other agents.
- *Decentralized learning*: which is also called *interactive learning*. In this category, more than one agent interact to execute the same learning process.

A multi-agent system may have several centralized learners trying to learn the same goal. Similarly a multi-agent system may have several decentralized agents trying to learn the same goal.

TABLE 4
How Big is an Exabyte? [84]

Kilobyte (KB)	2 Kilobytes: A Typewritten page. 100 Kilobytes: A low-resolution photograph.
Megabyte (MB)	1 Megabyte: A small novel OR a 3.5 inch floppy disk. 5 Megabytes: The complete works of Shakespeare. 10 Megabytes: A minute of high-fidelity sound.
Gigabyte (GB)	20 Gigabytes: A good collection of the works of Beethoven.
Terabytes (TB)	400 Terabytes: National Climactic Data Center (NOAA) database.
Petabytes (PB)	2 Petabytes: All U.S. academic research libraries.
Exabytes (EB)	5 Exabytes: All words ever spoken by human beings.

2.8 Agents on/for The Web

Today, the World Wide Web (WWW) is one of the most vital sources of information. Yet, there exists so much information on the Web, and most of it is so unorganized, that it is causing severe information overload. This section presents a survey of intelligent Web decision support systems, such as recommender systems that can help alleviate the problem of information overload, and discuss them within the context of the main paradigms in Artificial Intelligence, which include knowledge and reasoning, searching, planning, and learning.

In recent years, we have witnessed an explosive growth in the amount of information. Each day, more books and journals are published, more newspaper articles are written, more web pages are posted online, more office documents are prepared, more photos are taken, and more movies are created. According to two recent reports, 92 - 93% of newly created information is in digital form [66, 84]. In 1999, the worldwide production of original content, stored digitally using standard compression methods, was at least 1 terabyte/year for books, 2 terabytes/year for newspapers, 1 terabytes/year for periodicals and 19 terabytes/year for office documents [83]. In 2002, 5 exabytes of new information was produced [84], of which 4.6 exabytes of new information was in digital form. To get a feel of how big these amounts are, see Table 4. Table 5 gives an estimate of the size of the Internet.

According to studies and results given above, there were billions of documents on the World Wide Web (WWW), as of 2005. This over-abundance of information contributes to the reasons why we can get hundreds or even thousands of results for a simple search, and why we can find it hard to arrive at the resources that we need by wading through endless labyrinths of Web pages and

TABLE 5
The size of the Internet in terabytes in 2002. [84]

Surface Web	167
Deep Web	91,850
Email (originals)	440,606
Instant messaging	274
TOTAL	**532,897**

Websites. This problem is commonly referred to as *information overload.*

The Human brain is a fast and intelligent decision making organism, but when significant information overload is encountered, some additional external guidance may be needed. Systems that strive to achieve this aim may be known under different names such as decision support systems, recommender systems, customer relationship management systems, executive support systems, executive information systems, or personalized agents. We will refer to them as *Web decision support systems* throughout this chapter.

Most Web decision support systems rely on powerful techniques for sifting through large amounts of Web data in order to support decision making. The most powerful data sifting methods fall within the Web mining family of techniques, and these in turn tend to be classified into the following categories [70], [29], [70], [15]:

1. Web Content Mining (WCM): is concerned with mining knowledge from the actual *content* of Web pages.

2. Web Usage Mining (WUM): is concerned with mining knowledge from the user activity logs or access history data.

3. Web Structure Mining (WSM): is concerned with mining knowledge from the data that consists of the *hyper-link structure* between Web pages.

To date, information overload has been handled by several approaches, including

- **Information Retrieval and Query Modification**: Search engines try to deal with information overload by allowing the user to query for only the web pages that satisfy a specific information need [12], [126]. Most commercial server-based search engines[3] try to deal with information overload by building an index of the Web space beforehand, known as a *Web*

[3]http://searchenginewatch.com/showPage.html?page=2156221

index, and then performing simple Boolean matching of a user's query against this index [12], [126]. The Web index maps the occurrence of a term to the web pages where it is contained. Instead of searching an index, some client-based search agents will search the Web by following hyper-links to locate the most relevant documents. However, this approach may take too long and tends to suffer from low coverage. That is, some relevant web pages may never be reached by the search. Despite advances in Web search engines, the biggest obstacle toward finding relevant information is the inadequacy of user queries: Users often do not succeed in expressing their information needs in a way that optimizes retrieval. For this reason, *query modification* [134] has been proposed as one approach to improve users' queries by, for example adding more specific terms, in order to filter out irrelevant results.

- **User modeling**: *User Modeling* aims at identifying and representing the interests of a user. This in turn, allows to guide the user toward more relevant resources. The most well known challenge in user modeling is building a model of the users that is accurate and complete. For example, user profiles that are discovered through clustering the user records or user sessions on a website are one way to build aggregate user models [100], [91], [148]. However this approach requires powerful clustering algorithms that must not only search a large space of solutions to yield "accurate" partitions and cluster "representations", but also must be able to handle very large and high-dimensional data sets (such as clickstreams or user sessions). Among the techniques used for this purpose, we note several specialized evolutionary clustering algorithms, such as the Hierarchical Unsupervised Niche Clustering (HUNC), based on Genetic Algorithms [106], and Tracking Evolving Clusters in Noisy data Streams (TECNO-Streams), which is based on Artificial Immune Systems [96].

- **Information Filtering**: Information Filtering aims at sifting through many documents/Web pages in order to identify those that are relevant to a user. One type of information filtering is Personalized or Adaptive search that tailors the search process to a particular user [20], [19]. Hence the goal of information filtering is essentially to reduce the pool of information that the user will have to examine to a smaller subset. It can be done post-search, by classifying the results of a search so that those Web pages that do not fall within a user's interests can be weeded out. It can also be done in tandem with search, by incorporating the user's interests in a proactive manner as the search progresses in a personalized agent style.

- **Information Organization**: This task aims at changing the layout or structure of a large

amount of otherwise flatly organized documents or resources in such a way that the new structure (for example, a tree or hierarchy of documents) makes it easier for the users to locate relevant information [12], [126], [29], [20], [19]. A typical example of an organization of Web pages that albeit, was done by humans, is the *Yahoo directory*[4]. However, there is an increasing interest in performing such an organization automatically. For instance, a good organization can be obtained by clustering the original documents. One successful application of clustering can be found on certain search engine interfaces (e.g. Vivisimo[5]), where the results are clustered into separate categories instead of being simply listed. Presenting the results as separate subsets or clusters, tends to separate the results according to meaningful topics that help disambiguating documents with similar terms but different meaning.

- **Personalization**: aims at adapting a website to the user [94]. Personalization can be implemented in several different modes, including plain customization such as users configuring the layout and contents of the homepage (e.g. *MyYahoo*[6] or *NetVibes*[7] Web pages). However, the most powerful type of personalization may be automated recommendation systems that can adapt a Website or suggest items that are deemed to be relevant to a user, based on their recent activity. *Adaptive Websites* have been proposed as a challenge to the artificial intelligence community by Perkowitz and Etzioni [115], where websites can be built to "adapt" to their users automatically and hence become personalized and combat information overload. Several pioneering efforts in this direction include WebWatcher, Letizia, and Personal WebWatcher [10], [61], [76], [90]. Another even earlier effort to combat information overload was the Internet Softbot project [44], which has developed intelligent agents known as *softbots* that are based on planning, knowledge representation, and machine learning. Softbots rely on a metaphor viewing the Internet as an information food chain, and agents as information carnivores that must search for the relevant information on the Internet on behalf of a specific user.

In the following sections, we will focus on recommendation systems, since they represent the family of methods aiming to combat information overload, by adhering the most to Artificial Intelligence conventions. Figure 5 shows the basic modules of a typical personalization system.

[4] http://dir.yahoo.com
[5] http://vivisimo.com
[6] http://my.yahoo.com
[7] http://www.netvibes.com

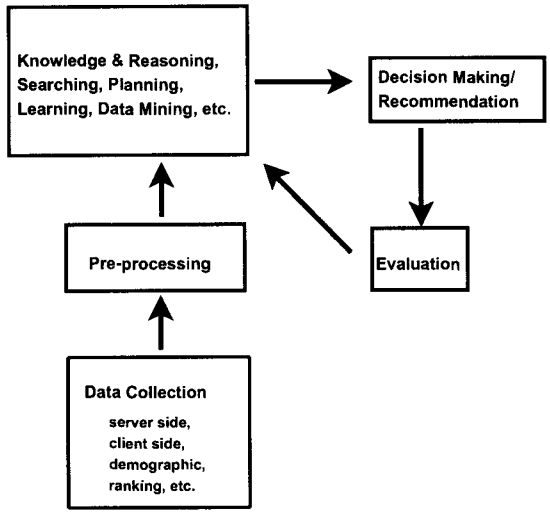


Figure 5. Modules and flow of typical recommendation systems.

2.8.1 Recommendation Systems According to User Granularity

In order to provide a personalized Web user experience that reduces information overload, most recommendation systems rely on user modeling as an important component. Elaine Rich provided a taxonomy of user models which have originated within the Human Computer Interaction field [125], and where user models can be classified into several dimensions: (a) individual versus group when dealing with the granularity of the user model, or (b) explicit versus implicit which concerns the way the model is extracted (i.e. with or without explicit input from the user or system designer), and (c) short versus long-term dimension when dealing with the scope of the user information over time. To classify Web decision support system models according to modeling the users *individually* or as a *group*, the former is called *individual user modeling* while the latter may be called *general access modeling*, *mass user modeling*, *group modeling* or *canonical modeling*. These two schemes are discussed in the rest of this section.

Group Modeling

Group modeling recommendation systems are systems that are based on models of groups. In general, the idea is to find the clusters representing different user profiles first, then create a strategy for each cluster, and finally classify a new user to one of the clusters and use the strategy generated for that cluster. Group modeling-based recommendation systems are mostly specialized for a specific website/domain. In this kind of system, the entire content of the website is known, and all the users' activities can be exploited by the system. Collaborative filtering systems [123],[56],

[46], [113] constitute the most common server side recommender systems.

Sources of data on a server-based group modeling recommender systems include visitor activity information, web server log files, web server instrumentation (plug-ins), TCP/IP packet sniffing (network collection), application server instrumentation, transactions, marketing programs (banner ads, emails, etc), demographic data (registration, third party overlay), supply chain (inventory and fulfillment), and other sources of data [17].

Advantages of server-side group modeling recommenders include the availability of a website index, query modification, and being able to combine and thus allow sharing of different users' interests.

Individual User Modeling

Individual user modeling-based recommender systems tend to be *personalized agents* [76], [90] that are specialized for a single user. Even though they can be located either on the server side or client side, the client side solution is more common and easier to implement. The idea is to learn the user behavior, their likes and dislikes, and produce recommendations. These systems can update themselves as the user's behavior changes. Input data for personal agents can be in many different formats: web click streams, rankings entered by the user, document files, emails, calendars, and many more... [17]

Personal assistants are mostly based on learning. Rankings can be used to guide supervised learning mechanisms [14]. Content-based filtering can also be used. Content-based approaches [16], [113] can be used alone or to assist collaborative filtering. Systems that use a combination of these two approaches [14] are called hybrid systems. Some advantages of individual modeling over group modeling include: taking into account the documents which are in use at the time of recommendation, and capturing the user's individual interests better. Query modification is also possible. Disadvantages of individual modeling systems include the unavailability of information about other users, and thus a stagnation on only the current user's past interests.

2.8.2 A Taxonomy of Recommendation Systems

Recommender systems can be classified in a variety of ways [133], but we adopt the following classification that depends on three main components defining the system:

1. *Past* information or data available to the recommender system *before* the recommendation time, such as user transactions, clickstreams, ratings, or demographic data,

2. *Current* or *active* user information inputs (such as which products were viewed/ purchased during the current session/transaction) at recommendation time,

3. *Algorithm* or procedure that combines the above two sources of data in order to come up with a list of recommended products.

When it comes to items (1) and (2) above, we notice that recommender systems can vary widely in their inputs Hence, depending on how the above three components are used and interrelated, automatic Web personalization can analyze the data to compute recommendations in different ways, including:

1. *Content-based or Item-based filtering*

2. *Collaborative filtering*

3. *Knowledge Engineering or Rule-based filtering*

4. *Demographic filtering*

5. *Hybrids*

These categories are discussed in the next sections. More information can be found in [113].

Content-based or Item-based Filtering

Content-based filtering systems recommend items to a given user, which are deemed to be similar to the items that the same user liked in the past or similar to the user profile in attributes. Item similarity is typically based on domain specific item attributes (such as author and subject for book items, artist and genre for music items), which are thus part of the items or user profile. Classical examples include Syskill and Webert [112], and Fab [13]. This approach has the advantage of easily including brand new items in the recommendation process, since there is no need for any previous implicit or explicit group user ratings or purchase data to make recommendations. Content-based filtering systems suffer from several limitations. First, they tend to be limited to certain types of content such as text and movies. Even in this case, the extracted features are limited in that they only capture certain aspects of the content. Second, they tend to provide over-specialized recommendations based only on user profiles or items that are similar to items previously rated by the user. Hence, users cannot explore new items that are different from those included in their profiles.

Collaborative Filtering

Based on the assumption that users with similar past behaviors (rating, browsing, or purchase history) have similar interests, a collaborative filtering system recommends items that are liked by other users with similar interests [132, 133]. This approach relies on a historic record of all user interests such as can be inferred from their ratings of the items on a website (products or web pages). Rating can be explicit (explicit ratings, previous purchases, customer satisfaction questionnaires) or implicit (browsing activity on a website or clickstreams). Typical examples include GroupLens [68][131] and a survey can be found in [132]. The recent Netflix price increased the attention on collaborative filtering[8][69]. Collaborative filtering can be either user-based or item-based. In user-based collaborative filtering, historic data such as purchases, visits, or ratings of items such as products or web pages, is used to form user neighborhoods of similar users. Later, for a new user, items are recommended if they are liked by this user's neighbors. In item-based collaborative filtering, historic data is used to form associations between items that tend to be liked by the same user. Later, for a new user with known ratings for a few items, other items that are associated with the known rated items are recommended. Computing recommendations can be based on lazy or eager learning to model the user interests. In lazy learning all previous user activities are simply stored, until recommendation time, when a new user is compared against all previous users to identify those who are similar, and in turn generate recommended items that are part of these similar users' interests. Lazy models are fast in training/learning, but they take up huge amounts of memory to store all user activities, and can be slow at recommendation time because of all the required comparisons. On the other hand, eager learning relies on data mining techniques to learn a summarized model of user interests (a decision tree, clusters/profiles, etc) that typically requires only a small fraction of the memory needed in lazy approaches. While eager learning can be slow, and is thus performed offline, using a learned model at recommendation time is generally much faster than lazy approaches. Collaborative filtering methods do not suffer from the over-specialized recommendations of content based filtering because they can suggest completely new types of items that are different from previously rated items, just because there are similar users who seem to be interested in these new items. Hence, they benefit from a surprise effect. However, they do suffer from a problem, known as the cold-start problem, which occurs when the system is faced with making recommendations to a new user that has not rated or purchased any item yet. Despite their limitations, collaborative filtering approaches exhibit a very desirable quality in that they are completely independent of the

[8] http://www.netflixprize.com

intrinsic properties of the items being rated or recommended. In particular, items that may be hard to describe using attributes, such as audio and images, as well as semantically rich text data can still be recommended based on latent similarities that are only captured through the social process of collaborative filtering.

Knowledge Engineering or Rule-based Filtering

In this approach, used frequently to customize products on e-commerce sites, the user answers several questions, until receiving a customized result such as a list of products or a custom-built configuration of a product (e.g. Dell's web page[9]). This approach is mostly based on heavy planning of a judicious set of questions and possible answer combinations by an expert, and establishing this dialog depends on manually coded scenarios that assume heavy knowledge about how each item fills the needs of a particular user.

Demographic Recommender Systems

In this approach, items are recommended to users based on their demographic attributes, such as gender, age, location, salary, etc. The recommendations can be based on handcrafted stereotypes derived from marketing research or on machine learning techniques [113] that learn to predict users' preferences from their demographic attributes. For instance, users can be classified into one of several classes based on their personal attributes, and this class information can form the basis for recommendations.

Hybrids

As we have already noted, each recommendation strategy has its own strengths and weaknesses. Hence it is not surprising that combining several recommendation strategies may provide better results than either strategy alone, as long as the combined scheme is able to tap on the strengths of the individual methods, while circumventing their individual weaknesses. There are several ways to categorize hybrid recommender systems [8, 113, 28, 92]. On a very general level, they can be classified into two general families depending on whether they combine several input data sources (e.g. user and item data) or several recommendation strategies (e.g. collaborative and content-based filtering) in order to form the recommended list.

[9] http://www.dell.com/

2.8.3 How Artificial Intelligence has been used for Web Recommendation Systems

The human brain can be quite fast at decision making, but cannot deal with the information overload problem encountered by users of the Web. Since a Web recommender system is expected to help with this problem, it is expected to be "intelligent". Without getting into a discussion of what being *intelligent* means for a recommender system, in this section we will discuss how basic artificial intelligence (AI) approaches have been applied for these systems. In particular, we will examine four approaches: Knowledge and Reasoning, Planning, Searching, and Learning.

Knowledge and Reasoning

Knowledge and reasoning constitute one class of artificial intelligence problem solving techniques [127]. In a logical knowledge-based problem solving system, there exists some knowledge about the problem domain, and logical reasoning is used to produce more knowledge and/or actions to perform. When compared to the Web decision support problem, it can be seen that this is one of the basic ideas behind recommender systems. The system has some knowledge about the user and/or items, such as likes, dislikes, similarity, being in the same shopping cart, etc; and the goal is to produce more information about likes, dislikes, similarity, etc. Consider content-based filtering as an example. Content-based filtering tries to generate a knowledge base which includes likes and dislikes of a user, using the similarity between items based on content. It assumes that anybody will like an item if it is similar to another item that they also liked. In other words, content-based filtering has the assumption that:

$-\forall i$, $Person_i$ *likes items that are similar to each other*

We need to have a knowledge base including the preferences of the user and the knowledge about similarities between items: The knowledge base might be of the form:

$-Person_i$ *likes* $item_k$.

$-Item_k$ *is similar to* $item_l$.

Then the inference will be:

$-IF$ $Person_i$ *likes* $item_k$ AND $Person_i$ *likes items that are similar to each other* AND $item_k$ *and* $item_l$ *are similar* $THEN$ $person_i$ *likes* $item_l$.

It is important to note, here, that assessing the "similarity" between items based on their content may be the most challenging task, and needs an accurate representation of content.

Let's present the problem in the context of collaborative filtering. In this case, we need to generate a knowledge base which includes which users have similar likes and dislikes, or who

likes/dislikes what. We need to have a knowledge base including the knowledge about similarities between people:

$-Person_i$ *and* $person_j$ *like similar sets of items.*

$-Person_j$ *likes* $item_k$.

Then the inference mechanism will work as follows:

$-IF\ Person_j\ likes\ item_k\ AND\ Person_i\ and\ Person_j\ like\ similar\ sets\ of\ items\ THEN\ Person_i\ likes\ item_k$.

As in content-based filtering, creating the part of the knowledge base that defines the "similarity" between users and the neighborhoods of similar users may be the most challenging aspect.

Another approach that uses Knowledge and Reasoning was presented in [95], where a knowledge base consisting of *fuzzy rules* was used to make an inference about the user's most likely interest. Each rule was built from a user profile, and the profiles were automatically discovered by clustering the user Web clickstream sessions. This approach proved to be as good as K-Nearest Neighbors based collaborative filtering, but worked with a knowledge base of very modest size compared to K-Nearest Neighbors.

A recent approach to enrich and automate efforts that counteract information overload is the Semantic Web. The Semantic Web is a project to create a universal medium for information exchange by putting documents with computer-processable meaning (semantics) on the World Wide Web [23, 1]. Most Semantic Web efforts so far have been based on knowledge representation efforts that try to create, encode, and make use of ontologies that offer one way to represent knowledge about facts and objects. For this purpose, several languages and conventions have been proposed to handle and process such ontologies, such as OWL and RDF to represent information, and how to reason with this knowledge, for example by making reasonable inference.

Searching

Another problem that is very central to AI is the search problem. Even though the WWW domain is a connected domain, because of its extraordinary size, search can become a huge challenge. There may be millions, or even billions of possibilities for the starting point and too many options to follow at each step. Moreover, there may be no unique definition for a final goal or an "optimum goal" other than the user's satisfaction, which is extremely subjective and dynamic. All commercial server-based search engines try to deal with this problem by building an index of the Web space beforehand, and then performing simple Boolean matching of a user's query against this index [12],

[126], [29], [20], [19]. Instead of searching an index, some client-based search agents will perform a graph style traversal or search of the Web by following hyper-links to locate the most relevant documents. However, this approach may take too long and tends to suffer from low coverage. Some relevant web pages may never be reached by the search.

Best-first Internet search is common in search engines. Search techniques including A* are mostly used while crawling [15]. In a client-based search algorithm called *fish search* [38, 39], the problem is searching for information on the WWW. The algorithm finds a starting point for the search, retrieves documents over the Internet, and scans for relevant information on the client side, then gets more links to continue searching. This is an example where the goodness of a hyper-link to follow is assessed before committing to follow it, for example by estimating its relevance score based on the existence of links between it and some other web pages with known relevance, or by estimating this score based on the "anchor text" that describes this hyper-link.

Another common search problem occurs when trying to create group user models by discovering the user profiles. Given Web server logs as input, in [104, 106, 97], a genetic algorithm based technique called *Hierarchical Niche Clustering* (HUNC) searches for user profiles and discovers associations between URL pages on a site . The input to the search consists of user sessions which are sets of URLs visited in each session. The output of the system consists of the set of URLs that represent groups of similar sessions (clusters) in an "optimal way". Generally speaking, there is no consensus on a unique definition for the "optimal representation". However, the most desirable solution is one where the ultimate cluster representatives (profiles) approximate the user sessions with precision and recall that are as high as possible [107]. This search problem also happens to be a clustering problem, that tries to find the clusters to represent the sessions in the "optimal way".

Recently, flocks of agents [116] and Ant Colony Optimization (ACO) [41, 73] methods have also been used for clustering. In the flock of agents approach, each agent represents one data item. The aim of the agents is to move with similar agents. They have simple local rules and evolve together in a 2D environment to accomplish this aim. The survey in [110] points to the striking similarities between the Web and natural environments, and gives an overview of applications of artificial life principles to searching and managing techniques, including an agent based approach called *InfoSpider*. Infospiders are adaptive agents that use a Genetic algorithm in their search strategy for *food* in the environment in order to optimize their *energy*. In addition to the above, more Evolutionary-based methods have found their use in Web information retrieval methods that are reviewed in [71].

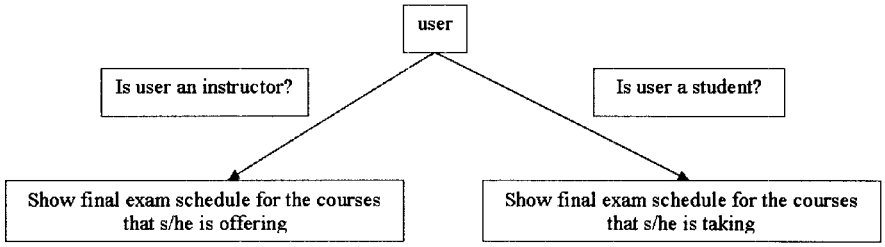


Figure 6. A simple example of a decision tree-based recommender system using a demographic attribute.

Planning

The third problem solving technique in AI is *planning*. In AI, planning is defined as solving the problem by forming a search for a solution over a space of plans, where beliefs about actions and their consequences are used [127]. When navigating the WWW, an *action* can be considered as following a specific link, purchasing a product, closing the connection, etc. The *state* may represent the current page, and may or may not include click history, demographic data, ratings, etc. Finally the *goal* is generally expected to be finding the needed information, making a desired purchase, entering a good rating, or visiting a specific link, etc. Even though planning is not considered to be a popular method for Web decision support systems, it is very commonly used in Expert Systems. An expert can define the actions, states, and goals as a recommendation strategy. However, as discussed in Section 2.8.2, since it heavily depends on human input, this approach may be cumbersome for large websites.

In Web decision support systems, one of the most common structures used in planning is decision trees. In a decision tree structure, each edge represents a question, and branches represent specific conditions satisfied by the attributes. Finally, leaf nodes can be considered as the final decision (e.g. class label). Figure 6 shows a simple example of a decision tree-based recommender system with demographic attributes, whereas Figure 7 presents a simple example of a decision tree-based Web decision support system with a content-based approach.

Decision trees may also be known under the names classification trees or reduction trees in the literature. In some approaches, leaf nodes of decision trees may have probabilistic results, such as a leaf with 3 classes, representing class a with probability p_a, class b with probability p_b and class c with probability $1 - (p_a + p_b)$. This adds a fuzziness to the decision mechanism. These kinds of trees are called *p*robabilistic decision trees and they are known to handle missing values [27]. When probability is considered, Bayesian belief networks (or just belief networks) also come to

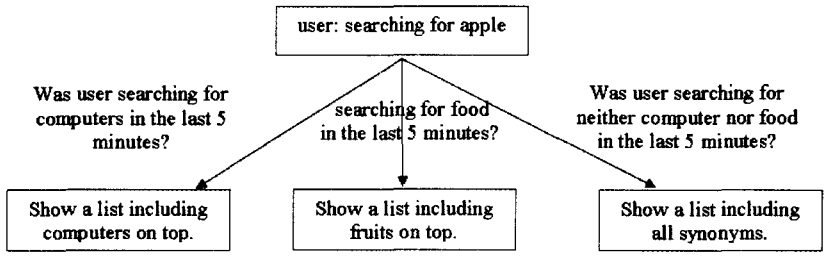


Figure 7. A simple example of a decision tree-based personalized search using a content-based approach.

mind [114]. Planning may be considered to be conceptually related to Knowledge-Engineering Web decision support system approaches.

Learning

In AI terminology, the idea behind *learning* is the ability to observe the results of interaction with the world and to use these observations to improve the decision making process [127]. Learning is very common in Web decision support systems. Whenever a system is able to deduce some *new* information about a user or a user profile, the system is said to be "learning". Two types of learning can be discussed: 1) Lazy and 2) Eager. In lazy learning, all the examples are kept and an answer to a query is produced by using the examples. No explicit model output is produced in lazy learning. However, in eager learning, training examples are used to generate a model output, and queries are answered by using this model output. A model output can be in the form of a tree, a set of rules, a function, a neural network, a set of cluster representatives, etc. Although lazy learners have low computational cost during training (which is close to zero), they require high storage (memory) cost and computational cost at decision time. Lazy learning is more resistant to noisy data and responds better to dynamic data. On the other hand, when the data set changes, one may need to rebuild an eager learning model output. K-Nearest Neighbors is one of the most popular lazy learning algorithms, while neural networks and decision trees are examples of eager learning methods. *Syskill and Webert* [111] is an early example of a Web decision support system, that achieved information filtering using several lazy and eager approaches including K-Nearest Neighbors, Bayesian classifiers, and Decision Trees to learn which Web pages are interesting to a particular user. The work in [24] presents a Neural Network approach to learn the interests of a user from the previous ratings of similar users. This approach falls within the realm of collaborative filtering recommender systems.

Below, we will discuss some of these approaches. We start with decision trees that were

also discussed in Section 2.8.3. In working with a decision tree, there are two stages: 1) Forming or learning the tree, and 2) Applying the tree on new inputs to make predictions. The most common method for forming the tree is learning. ID3, C4.5, C5.0, CART are the most common algorithms to generate/learn decision trees [121]. There are two ways to use decision trees in recommender systems: 1) representing the recommender system as a decision tree, and 2) using decision trees for determining the important attributes, as is used in text categorization. An example of text categorization using decision trees can be found in [111] and [75].

Neural Networks (NNs) constitute a learning paradigm inspired by the biological nervous system [127, 54]. In intelligent decision support systems, NNs can be used for classifying users or items, clustering users or items, predicting a user's next move, adaptive filtering [20, 19], or user modeling [103]. Most of the NNs are used for supervised learning. However, self-organizing maps have been used for clustering within an unsupervised learning framework. An application of self-organizing maps for the visual exploration of the Internet is WEBSOM [67, 58]. In the system, similar documents are located near each other on a two-dimensional map.

Reinforcement learning [135] is mostly common in user or client-based support systems, frequently known as "personal agents". Rankings, ratings or clicks made by the user are used as reward and punishment mechanisms [61]. *WebWatcher* is a good example of a learning-based personalization system, that acts like a learning tour guide for the WWW [10, 61]. When the user is browsing the web, the WebWatcher agent accompanies the user and highlights the suggested hyper-links, while learning from its experience in order to improve its advice-giving skills. There are three approaches used in learning: 1) Learning from previous tours, 2) Learning from hypertext structure, and 3) Combination of the first two. *Learning from previous tours* is like a combination of content-based filtering and collaborative filtering with learning. According to the user interest -which are the keywords that the user typed at the beginning of the tour- the system gets the most related links from the current page. This can be considered as an application of content-based filtering. On the other hand, the definitions (or annotations) of the hyper-links are enriched (in other words learned) by the user interest (i.e. keywords typed) when the user follows the hyper-link. Later, this knowledge is used for advising other users with similar interests. This idea adheres to the principles of collaborative filtering. The second learning approach, learning from hypertext structure, is based on reinforcement learning. In this method, a hyperlink is augmented using the words encountered in the pages downstream from it. Q-learning is used with states corresponding to Web pages and actions corresponding to hyper-links. The reward function, in this case, is a kind

TABLE 6
Examples of Intelligent Personalization Systems, their user granularity, and their A.I. paradigm

Research/System	Main Author Year	Application	Input Data	Recommendation Paradigm	User Granularity	AI Task
Letizia [76]	Lieberman 1995	Behavior-based User interface agent that assists browsing	User's browsing behavior	Content-based	Individual	Learning and Search
WebWatcher [10], [61]	Armstrong 1995 Joachims 1997	Learning tour guide which assists browsing	Users' browsing behavior	Content-based and Collaborative	Group	Learning
Personal WebWatcher [90]	Mladenic 1996	Learning agent that assists browsing	Users' browsing behavior	Content-based	Individual	Learning (Text)
Syskill & Webert [111], [112]	Pazzani 1996	A software agent that learns to rate pages on the WWW	User's ratings	Content-based	Individual	Learning
Fab [14],[13]	Balabanovic 1997	Hybrid, adaptive recommendation system for browsing	User's browsing behavior and user rankings	Content-based and Collaborative	Individual and Group	Learning Search Heuristics

of similarity between the Web page and the user interests or keywords. Table 6 lists some examples of intelligent recommender/personalization systems, their user granularity, and their A.I. paradigm.

2.8.4 Evaluating The Performance of Web Recommender Systems

Evaluating a recommender system can be nearly as hard as designing and implementing the system, in part because no simple, objective, and general agreed upon mathematical formula is always available to measure success [12, 126, 20, 19]. One problem suffered by some systems is *over-specialization.* When the recommendations are limited to the user's behavior or user's profile, the user can be restricted to seeing only similar items, and there will be no randomness. In AI, this problem is known as, the exploration/exploitation dilemma. Collaborative filtering can counteract over-specialization by suggesting different items.

Evaluating information retrieval systems can be done if one has available, a set of user queries and a labeled set of search results (relevant and non-relevant). In this case, *precision* (proportion of retrieved items that are really relevant) and *recall/coverage* (proportion of all items, known to be relevant, that are retrieved) are typically used as goodness metrics [126, 12]. One method for evaluating a Web recommender system is asking for a ranking or a rating of the results from the users. However, this can be subjective. Moreover, if the study is for research purposes, it can be hard to find a sufficient number of real users with diverse interests for the experiment. For this reason, historical data has also been used in research studies. In this case, the output of a Web recommender system is compared to the real moves of the user in the historical data, and metrics such as precision and coverage are computed [126, 12]. One popular way to assess the success of a system is to compare

TABLE 7
Evaluation examples

Research/System	Evaluation Style
Letizia (Lieberman, 1995 [76])	No evaluation was given in [76]
Webwatcher (Armstrong, 1995 Joachims, 1997 [10], [61])	1) Random, Popularity Match, Annotate, RL suggestions are compared according to accuracy 2) Random, Annotate, Human suggestions are compared according to accuracy
Personal WebWatcher (Mladenic, 1996 [90])	Precision (1, if a correct suggestion; 0 else) and percent of correctly classified examples with different parameters.
Syskill & Webert (Pazzani, 1996 [111], [112])	Accuracy computation using user ratings with different algorithms like Nearest Neighbor, PEBLS, ID3, Rocchio's algorithms, neural nets and Bayesian classifier for different domains.
Fab (Balabanovic, 1997 [14],[13])	Predict user's rankings and compute the distance according to ndpm measure([149])

it, for example with recommending the default, most popular, or even a randomly selected item. In Fab [14, 13], a performance evaluation based on ranking was presented. Although evaluating the success of a recommendation is subjective, there may exist formulas for defining the users or items. Some systems create profiles, then generate recommendations using these profiles [95, 103]. Therefore, evaluating the accuracy of the profiling may give an early evaluation of the Web decision support system. The discovered profiles can also be considered as frequent itemsets, or patterns, and provide one way to form a summary of the input data. As a summary, profiles represent a reduced form of the data that is at the same time, as close as possible to the original input data. This description is reminiscent of an information retrieval scenario, in the sense that profiles that are retrieved should be as close as possible to the original session data. Closeness should take into account both (i) precision (a summary profile's items are all correct or included in the original input data, i.e. they include only the true data items) and (ii) coverage/recall (a summary profile's items are complete compared to the data that is summarized, i.e. they include all the data items). Table 7 presents examples of evaluations used with some Web decision support systems.

2.8.5 Summary of Agents on/for the Web

We have presented an overview of methods and paradigms used in Web recommender systems, which are indispensable to counteract the severe information overload problems encountered

by users of the Web. Most of these systems try to counteract information overload by resorting to some way to sift through large amounts of Web data, such as by Web mining, and can provide a solution through Web search, user modeling, information filtering, information organization, or personalization. When taking into account the user-based granularity of these systems, we can distinguish between group and individual user modeling, with each level of granularity having its own strengths and weaknesses. When dealing with huge populations of users, the individual level that may be optimal and faithful to reality from a knowledge representation point of view, may not even be feasible or practical to work with from a computational and memory storage point of view. This is where group user modeling is needed. We have focused on recommender systems as the type of Web decision support systems that adheres the most to the principles and expectations of an Artificial Intelligence paradigm, and have explained how the main A.I. paradigms have been embodied within these systems. In particular, we have elaborated on Web recommender or personalization systems that rely on Knowledge and Reasoning, Searching, Planning, and Learning. Even though most commercial systems prefer group modeling, individual modeling is becoming more popular these days. Increasing personalized searches, requiring users to login on some websites, and personalized news are all first and basic steps of these moves. However, they may conflict with the user's expectations of privacy. Although, the importance of Web decision support systems is obvious, profiling, defining, and recognizing users is not easy. On the WWW, both users' behavior and the Internet structure and content are highly dynamic. Therefore, systems must be able to function in an online and realtime setting for the best effectiveness.

Finally, to take a glimpse toward the future, we must note that with the advent of the Web 2.0, which is characterized by the proliferation of *social and collaborative networks* (e.g. *Facebook, LinkedIn*), *folksonomies* that include tagging and sharing bookmarks and content (e.g. Flickr and del.icio.us), and collaborative *user generated content* (e.g. Wikipedia, Wikimapia, Digg, Amazon reviews, Blogs, etc.), the Web infosphere is taking a significantly different shape. Another phenomenon that will contribute to reducing information overload is the advent of the semantic Web. One of the most desirable outcomes of Web 2.0 is an emerging *self-organization* and user-driven distributed information enrichment (e.g. *tagging*) of the Web. Social and collaborative networks and folksonomies have not only led to the *creation* of *new types of data* that will challenge all efforts in analyzing and exploiting such data, but also provided an additional *layer of organization and implicit knowledge* that may hold even more power against information overload.

CHAPTER 3

IMPROVED FLOCKS OF AGENTS-BASED CLUSTERING

In this Chapter we describe our proposed improvements to flocks of agents based static clustering. We first discuss automated parameter estimation in Section 3.1.1. Then in Section 3.1.2, we present an annealing approach for adaptive thresholding. in Section 3.2.1 and 3.2.2, the new (K-means+FClust) and (SPKM+FClust) Hybrid algorithms are presented respectively. We also validate each development using experiments in Section 3.3.

3.1 Improved Distance Threshold Estimates

Setting the distance threshold parameter in FClust plays a big role in defining the quality of results and convergence of FClust. In fact, most clustering algorithms crucially depend on a similar parameter either directly or indirectly. In this section, we present an improved way to automatically set this parameter in FClust.

3.1.1 Alternative Fixed Thresholding

In our experiments, we observed that the original FClust was not suitable for some very high dimensional sparse datasets, such as Web usage data. Therefore, we had to find a new formulation for the similarity threshold as given by Equation (26). When the maximum similarity is very high and the average similarity very low, the similarity threshold computed by (24) is too high for FClust to converge. Equation (26) includes a flexible way to tune the similarity threshold, using a tuning factor α, where $1 \leq \alpha \leq \frac{sim_{max}}{sim_{average}}$.

$$sim_{th} = \alpha \times sim_{average} \tag{26}$$

Another problem observed is that the ideal distance computation results in an infinite value via Equation (23) if many similarity values are 1, because $sim_{th} = 1$. Equation (27) proposes an alternative formulation, which can handle many 1-similarities.

$$d_{ideal}(i,j) = \begin{cases} \frac{1-sim(i,j)}{1-sim_{th}} \times d_{th}, & sim_{th} \neq 1 \\ 0, & sim_{th} = 1 \end{cases} \quad (27)$$

Additionally, when the number of data records is small, setting the distance threshold is very hard because, if it is too low, the agents cannot meet each other, while if it is too high, the ideal distances computed via Equations (23) and (27) will be too high, and cluster formation will not occur. As a solution, a new parameter for ideal distance tuning is added, as given in Equation (28). The ideal distance threshold constant, d_{ideal_th}, used in Equation (28), is also used, replacing d_{th} during the cluster formation given in Algorithm 9, line 5.

$$d_{ideal}(i,j) = \begin{cases} \frac{1-sim(i,j)}{1-sim_{th}} \times d_{ideal\_th}, & sim_{th} \neq 1 \\ 0, & sim_{th} = 1 \end{cases} \quad (28)$$

In addition to these simple tuning modifications, an annealing version of FClust is proposed in Section 3.1.2.

3.1.2 Adaptive Thresholding using FClust Annealing

The distance threshold, d_{th}, has a great effect on the convergence of clustering. When d_{th} is too high, too many agents affect a given agent. This results in an attempt to satisfy too many constraints at once, and the algorithm does not converge. However, when d_{th} is too low, the neighborhood tends to be too narrow for agents to see each other. Therefore they may not affect each other. Even when convergence is possible, it takes too many iterations.

One way to decrease the sensitivity to a fixed threshold is by using an annealing schedule for d_{th}, where d_{th} is highest at the beginning and then decreases as the number of iterations increases. In that case, d_{ideal} is computed using Equation (28). A possible cooling schedule for d_{th} is given in Equation (29) and shown in Figure 8.

$$d_{th}(t) = \frac{d_{th}(0) - d_{th}(N)}{cosh(\frac{10 \times t}{N})} + d_{th}(N) \quad (29)$$

where, t is the iteration number and N is the number of iterations. d_{th} starts from $d_{th}(0)$ and decreases down to $d_{th}(N)$.

During the cluster formation process, in line 5 of Algorithm 9, d_{ideal_th} is used to define the neighborhood.

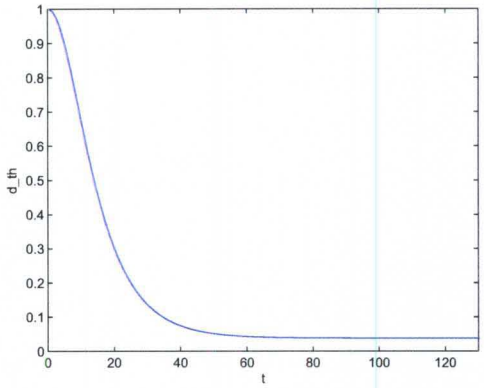


Figure 8. Cooling Schedule for d_{th} for Annealing FClust.

3.2 Hybrid Algorithms

K-means and SPKM are fast algorithms with $O(n)$ time complexity. However, the number of expected clusters, K, needs to be given as an input to the K-means and SPKM algorithms. Additionally, K-means and SPKM may not cluster the data successfully if the cluster boundaries are not hyper-spherical. Unlike K-means and SPKM, FClust has an adaptive way to extract a reasonable number of clusters without any boundary restrictions. However, the complexity cost for FClust is higher than K-means and SPKM. In this section, we propose the (K-means+FClust) and (SPKM+FClust) Hybrid Algorithms which aim to benefit from the advantages of K-means, SPKM, and FClust.

3.2.1 The (K-means+FClust) Hybrid Algorithm

K-means is a fast algorithm with $O(n)$ time complexity. However, the number of expected clusters, K, needs to be given as an input to the K-means algorithm. Moreover, K-means can only find clusters with hyperspherical shapes. Therefore, if the clusters are not hyperspherical, K-means may not estimate the cluster boundaries correctly. A hybrid algorithm of K-means and FClust is presented in Algorithm 10 to take advantage of the speed of K-means while also benefiting from the power of automatically determining the number of clusters in FClust. In the hybrid algorithm, initially, K-means is run with a high number of clusters which is more than the expected number of clusters. The cluster centroids are then extracted to get cluster representatives, and these representatives are mapped to the agent domain. Next, FClust is run on this smaller number of agents (relative to the size of the input data set) and the clustering results are mapped back to the input data domain. Note that, in the hybrid approach, each agent is mapped to the closest

group of data records via its group centroid, since each agent represents a group centroid. The time complexity for K-means, lines 1 to 2 in Algorithm 10, is $O(n)$, and the time complexity of the FClust part, in lines 3 to 4, is $O(K^2)$. Since the number of agents K is very small compared to the number of agents n, as long as $K \leq \sqrt{n}$ (which is always the case in practice), the time complexity and memory complexity of the (K-means+FClust) hybrid is $O(n)$. Therefore the hybrid version reduces the *time and memory* complexities *from quadratic to linear.*

Algorithm 10 (K-means+FClust) Hybrid Algorithm

Input: Dataset with n data records, an over-specified number of initial clusters, $K \leq \sqrt{n}$.
Output: Visualization of interaction between the data items and agent-cluster formation.

1: Run K-means on the original dataset with K clusters, where $K \leq \sqrt{n}$).
2: Extract K cluster centroids from K-means' results,
3: Map each cluster centroid to an agent for FClust,
4: Run FClust with K agents.

Advantages of the (K-means+FClust) Hybrid Algorithm

The advantages of the (K-means+FClust) Hybrid algorithm can be listed as:

- Linear complexity: Unlike FClust, the (FClust+K-means) Hybrid algorithm has linear complexity inherited from K-means.
- Unlike K-means, the (FClust+K-means) Hybrid algorithm has an adaptive way to extract a reasonable number of clusters.
- Unlike K-means, the (FClust+K-means) Hybrid algorithm suffers from no hyperspherical boundary restrictions on the extracted clusters.

Stopping Criterion

During the movement of the agents, the goal of each agent is to move such that the agent will be located at an ideal distance to all neighboring agents. Therefore, the procedure can be considered as a constraint satisfaction problem [145], where having the ideal distance between every agent pair represents the optimal goal. The difference between the real distance between two agents and their ideal distance is called the *ideal distance error* in this document as shown in Equation (30). One proposed stopping criterion is to halt if the difference in total ideal distances between all agent pairs in two consecutive iterations is small enough. Since this stopping criterion is feasible

TABLE 8

Comparison of different stopping criteria.

Algorithm	Advantage	Disadvantage
Human experts [120, 37, 117]	Takes advantage of human intelligence.	Since not automated, more costly and error prone.
Spatial Entropy-based[117]	Automated. Has lower computational complexity than ideal distance-based stopping.	Can stop early, thus separate some clusters. Harder implementation. Experimentally set parameters may not work for other datasets.
Ideal distance-based	Automated. Serves the constraint satisfaction nature of FClust. Easy implementation.	High computational complexity $O(n^2)$, thus applicable only on small datasets.

when the number of agents is small, it will be used for the (K-means+FClust) Hybrid algorithm in the experiments of Section 3.3.3.

$$ideal\ distance\ error(i,j) = |d_{ideal}(i,j) - d(i,j)| \tag{30}$$

Table 8 presents a comparison of this stopping criterion and the stopping criteria presented in Section 2.5.2.

3.2.2 The (SPKM+FClust) Hybrid Algorithm

SPKM is a fast algorithm with *O(n)* time complexity for clustering high dimensional and sparse data sets. However, the number of expected clusters, K, needs to be given as an input to SPKM. A hybrid of SPKM and FClust is presented in Algorithm 11 to take advantage of the speed of SPKM while also benefiting from FClust's aptitude to automatically determine the number of clusters. In the hybrid algorithm, initially SPKM is run with a high number of clusters K which is more than the expected number of clusters. The cluster centroids are then extracted to get a representative of the data from each cluster, and *only these cluster representatives* are mapped to the agent domain. Next, FClust is run on this smaller number of agents (relative to the size of the input data set) and the clustering results are mapped back to the input data domain. Note that, in the hybrid approach, each agent is mapped to the closest group of data records via its group centroid, since each agent represents a group centroid. The time complexity for SPKM, lines 1 to 2 in Algorithm 11, is *O(n)*, and the time complexity of the FClust part, in lines 3 to 4, is $O(K^2)$. Since the number of agents K is very small compared to the number of agents n, as long as $K \leq \sqrt{n}$ (which is always the case in practice), the time complexity and memory complexity of

the (SPKM+FClust) hybrid is $O(n)$. Therefore the hybrid version reduces the *time and memory* complexities *from quadratic to linear.*

Algorithm 11 (SPKM+FClust) Hybrid Algorithm

Input: Data set $\chi \in \Re^d$ where $|\chi|=n$; An over-specified number of initial clusters, $K \leq \sqrt{n}$.
Output: Visualization of interaction between the data items and agent-cluster formation.

1: Run SPKM on the original data set with K clusters, where $K \leq \sqrt{n}$).
2: Extract K cluster centroids from SPKM' results,
3: Map each cluster centroid to an agent for FClust,
4: Run FClust with K agents.

Stopping Criterion

The (SPKM+FClust) hybrid algorithm uses two different stopping criteria, one for SPKM and one for FClust. For SPKM, the most common criterion is tracing the increase in the total similarity, as mentioned in Section 2.1.3. For FClust the most common method for stopping the algorithm is using human experts [120, 37, 117]. An expert keeps watching the visualization panel until stable clusters are formed. At that time, the algorithm is stopped. However, an automated method was also presented in [117], which used the spatial entropy of the agents relative to their location on a grid imposed on the agent space. The problem with this criterion is that the entropy may remain unchanged for a long while whenever new neighbors do not met, but after a meeting occurs, changes may start re-occurring. Thus, the above stopping criterion cannot capture these delayed dynamics, and the algorithm risks to be stopped before convergence. During the movement of the agents, the goal of each agent is to move such that the agent will be located at an ideal distance to all neighboring agents. Therefore, the procedure can be considered as a constraint satisfaction problem where having the ideal distance between every agent pair represents the optimal goal. The difference between the real distance between two agents and their ideal distance is called the ideal distance error in this document as shown in (30). One proposed stopping criterion is to halt if the difference in total ideal distances between all agent pairs in two consecutive iterations is small enough. Since this stopping criterion is feasible when the number of agents is small, it will be used for the (SPKM+FClust) Hybrid algorithm in the experiments of Section 3.3.3.

TABLE 9
Datasets.

Dataset ID	Number of Items	Number of Attributes	Number of Clusters	Maximum Similarity	Average Similarity
I	1600	2	2	0.999948	0.814345
II	811	2	3	0.997500	0.732619
WebM	1704 (sessions)	343 (urls)	NA	1.000000	0.059718
Iris	150	4	3	1.0	0.709334
Pima	768	8	2	0.986171	0.832665

3.3 Experimental Results

In this section, we describe our experiments and their results for FClust, FClust-Annealing, (K-means+FClust) Hybrid, and (SPKM+FClust) Hybrid. We start by describing the datasets that we will use in our experiments, in Section 3.3.1. Then we proceed to explaining how post processing is applied to extract clusters in Section 3.3.2. Finally, Section 3.3.3 presents the experimental results observed for FClust, FClust-Annealing, (K-means+FClust) Hybrid, and (SPKM+FClust) Hybrid on different datasets. No experiments are presented for (K-means+FClust) and (SPKM+FClust) Hybrid-Annealing because, the aim behind annealing is to have a bigger neighborhood initially and reducing it in time to speed the convergence of clusters. However, in the hybrid versions, since the number of agents is already small, a bigger distance threshold, thus a wider neighborhood size, is being used, and naturally convergence is very fast. Therefore, annealing is not used for the hybrid experiments.

3.3.1 Datasets

As shown in Table 9, datasets I and II are synthetic datasets, whereas dataset WebM consists of real Web usage sessions. Iris and Pima are also real life datasets from the UCI machine learning repository [1]. Datasets I and II have 2 attributes and are thus suitable to show the clustering results visually. To compare the proposed improvements and the (K-Means+FClust) hybrid algorithm with the original FClust algorithm, datasets Iris and Pima are also used.

Dataset WebM, the Web usage data, consists of Web usage *sessions* logged by the server of the CECS Department at the University of Missouri-Columbia (considered as a benchmark data set) based on the pre-processing method and results in [98]. Each session is a set of *URLs* that were visited during that session. Since each URL or item can be considered as one dimension, this results

[1] http://archive.ics.uci.edu/ml/

in a huge dimensionality. In the dataset, there exist 1704 sessions and 343 URLs, representing the number of data records and attributes, respectively. The maximum similarity between sessions is 1.00 and the average similarity is 0.06.

In the experiments, two different similarity measures are used, the Manhattan based similarity for datasets I and II, and the cosine similarity for dataset WebM. The Manhattan based similarity is used for the linearly normalized Iris and Pima datasets.

The Manhattan based (L1) similarity of two data records x_i and x_j is given by Equation 4. When the data is linearly normalized to $[0, 1]$, the Manhattan based similarity is the same as the 1-norm similarity, which is used in (4).

Given that $\overline{s_i}$ and $\overline{s_j}$ are two sessions, each formed of a list of $|\overline{s_i}|$ and $|\overline{s_j}|$ URLs visited in each user session respectively, the cosine similarity is computed as follows:

$$sim(\overline{s_i}, \overline{s_j}) = \frac{|\overline{s_i} \bigcap \overline{s_j}|}{\sqrt{|\overline{s_i}| \times |\overline{s_j}|}} \tag{31}$$

3.3.2 Post Processing

After the agent clusters are formed, a post-processing phase is needed to cluster the data and validate the results. Post-processing depends on the data properties. If the data has 3 attributes or less, the data points are plotted with different colors depending on the cluster assigned. If the data has a class attribute, then the cluster error, given in Algorithm 3, is also computed.

Algorithm 12 Synthetic Data Post-Processing Algorithm

Input: Agents' coordinates and their cluster labels, Original data set, S_{TH}: minimum cluster size
Output: Clustered data, plot of agents and data set colored according to their cluster label.

1: Create as many clusters as formed for the agents.
2: Label each data record with the same label as the agent representing that data record.
3: Keep only the clusters with enough data records (i.e. size is above S_{TH}).
4: Plot agents on the visualization panel, colored according to cluster label.
5: If the data is in 2D or 3D, then plot the data records, colored according to the cluster label (for validation).
6: If the data has a class label, compute the cluster error via Algorithm 3.

Moreover, if the data consists of Web user sessions, then profiles are extracted as shown in Algorithm 13, line 6. In Algorithm 12, line 3, Algorithm 13, line 3, Algorithm 14, line 5, and Algorithm 15, line 4, S_{TH}, denotes the *session threshold*, i.e. the minimum number of sessions required for a cluster to be valid. If a data record is a bag of items, as in the case of Web usage data, the value of item_count_threshold used in Algorithm 13, line 6 is given in Equation (32), where $ICTF$ denotes the *Item Count Threshold Frequency*, where $0 \leq ICTF \leq 1$ is a real number. As a

Algorithm 13 Web Usage Data Post-Processing Algorithm

Input: Agents' coordinates and their cluster labels, Original data set, S_{TH}: minimum cluster size
Output: Clustered data, plot of agents colored according to their cluster label, and user profiles.

1: Create as many clusters as formed for the agents
2: Label each session with the same label as the agent representing that session.
3: Keep the clusters with more than S_{TH} sessions.
4: Plot agents on the visualization panel, colored according to their cluster label.
5: **for all** Clusters **do**
6: Find the URLs which are visited more than *item_count_threshold* in all the sessions of that cluster.
7: **end for**

result, each set of URLs, extracted in line 6 of Algorithm 13, can be considered as a pattern that represents a Web user profile that summarizes the sessions assigned to that cluster.

$$item\_count\_threshold = ICTF * cluster\_size \tag{32}$$

Algorithm 14 (K-Means+FClust) Post-Processing Algorithm

Input: Cluster means produced by K-means, agents' coordinates and cluster labels from FClust's output.
Output: Clustered data, and plot of agents and dataset, colored according to their cluster label.

1: Create as many clusters as formed for the agents
2: Map data records to the cluster means of K-means, in other word to agents.
3: If data is in 2D or 3D, plot data records, colored according to their cluster labels from K-means.
4: Label each data record with the same label as the FClust-generated label of the agent representing that data record.
5: Keep the clusters with more data records than S_{TH}.
6: Plot agents on the visualization panel, colored according to their cluster label.
7: If data is in 2D or 3D, plot data records, colored according to their cluster label from FClust.
8: If the data has a class label, compute the cluster error via Algorithm 3.

Algorithm 15 (SPKM+FClust) Web Usage Data Post-Processing.

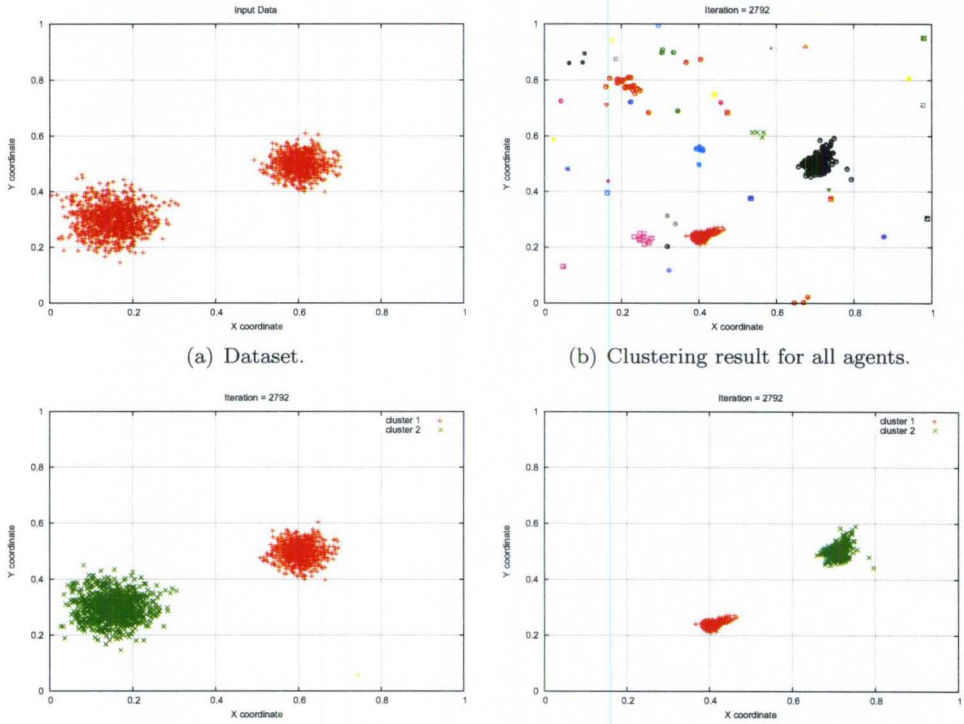
Input: Agents' coordinates and their cluster labels from FClust output, Original data set, S_{TH}: minimum cluster size, Data cluster labels from SPKM output, URL list.
Output: Clustered data, plot of agents colored according to their cluster label, and extracted user profiles as sets of URLs.

1: Create as many clusters as formed for the agents using Algorithm 9.
2: Map data records to the cluster centroids of SPKM, in other words to agents.
3: Label each session with the same label as the FClust-generated label of the agent representing that session.
4: Keep the clusters with more than S_{TH} sessions.
5: Plot agents on the visualization panel, colored according to their cluster label.
6: **for all** Clusters **do**
7: Find the URLs which are visited more than *item_ count_threshold* in all the sessions of that cluster.
8: **end for**

After the (SPKM+FClust) hybrid algorithm , Algorithm 9 is used to form agent clusters. Then, Web session clusters are formed and cluster profiles are generated using Algorithm 15, where each *profile* is represented by a URL set. To evaluate and compare the quality of clusters, the average inter-cluster (or between-cluster) and intra-cluster (or within-cluster) similarities are computed. The similarity of any cluster to an empty cluster (in the case of evaluating the results before post-processing) is assumed to be 0.

3.3.3 Results

In Sections 3.3.3, we start with 2D datasets to allow us to do a visual evaluation. Then we proceed with the Web usage data as a challenging, high dimensional, real life data example. Finally, we present our results for the Iris and Pima datasets and compare the clustering outputs to the data class provided as part of the datasets.



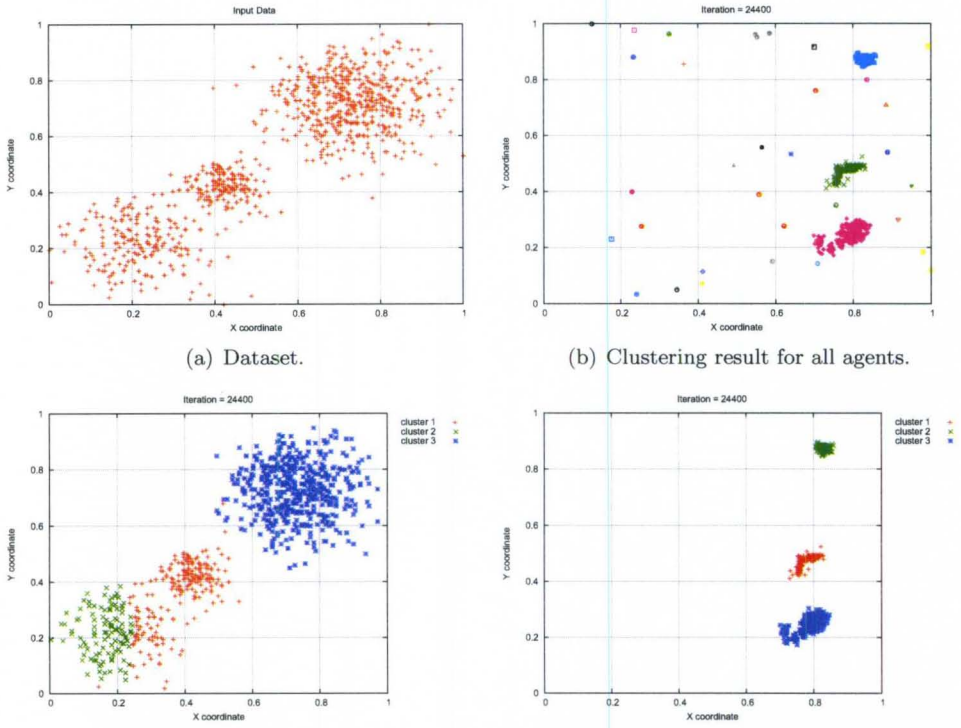
(a) Dataset.

(b) Clustering result for all agents.

(c) Clustering result for dataset.

(d) Agent clusters generated after post processing and assigning agents to clusters.

Figure 9. Clustering a dataset with two clusters using FClust where d_{th}=0.04, sim_{th}=0.91.



(a) Dataset.

(b) Clustering result for all agents.

(c) Clustering result for dataset.

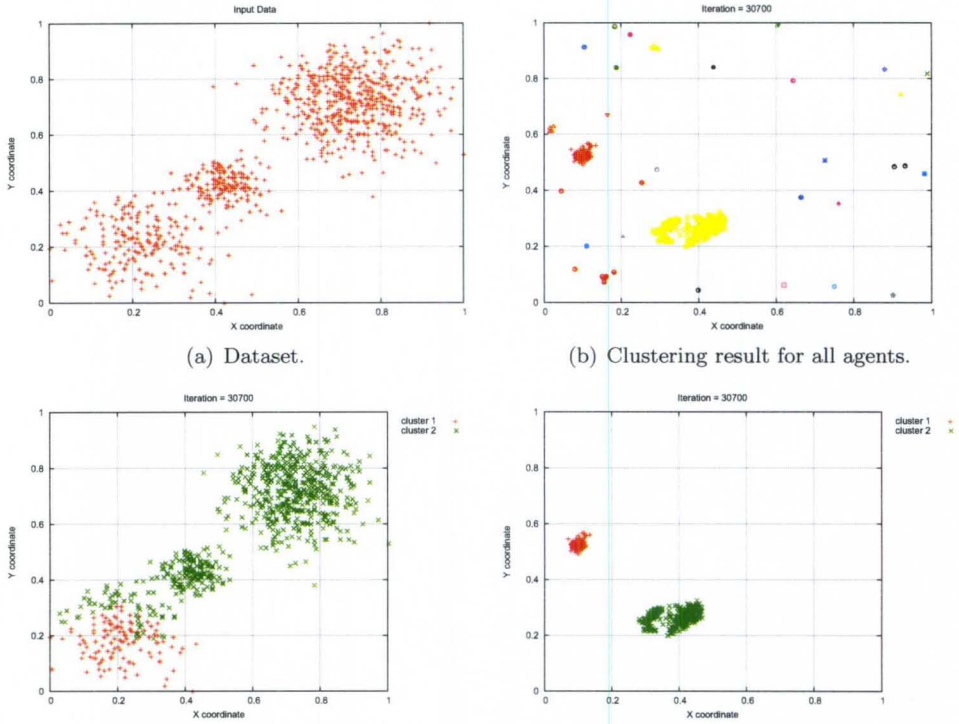
(d) Agent clusters generated after post processing and assigning agents to clusters.

Figure 10. Clustering a dataset with three clusters using FClust at iteration 24400.

FClust Results for 2D and Web Usage Datasets

Figure 9(a) shows an example of a data set with two clusters, and the resulting agents space are shown in Figure 9(b). In Figure 9(d), agent clusters which include more than S_{TH} data records are shown. And in Figure 9(c), the clustered data is shown. Figure 10 shows the clustering result for a more complicated data set, given in Figure 10(a), using the Manhattan similarity given in Equation (4), and d_{th}=0.04, sim_{th}=0.86. Likewise, Figure 10(d) is the post-processed version of Figure 10(b). When we compare the results in Figure 9(c) and Figure 10(c), we observe that, when the data clusters are not strictly separated, the cluster formation algorithm, Algorithm 9, may suffer from a bridging effect that results in merging two distinct clusters. Note that, since the synthetic datasets used in our experiments already had attributes between 0 and 1, we did not linearly normalize them in [0,1].

Figure 11 shows the results after more iterations compared to Figure 10 ($d_{th} = 0.04$, $sim_{th} = 0.86$). This shows that, if the agents' movement is stopped in a wrong state, different clusters may be assigned to the same cluster. Therefore, the stopping criteria is crucially important for overlapping data sets.



(a) Dataset.

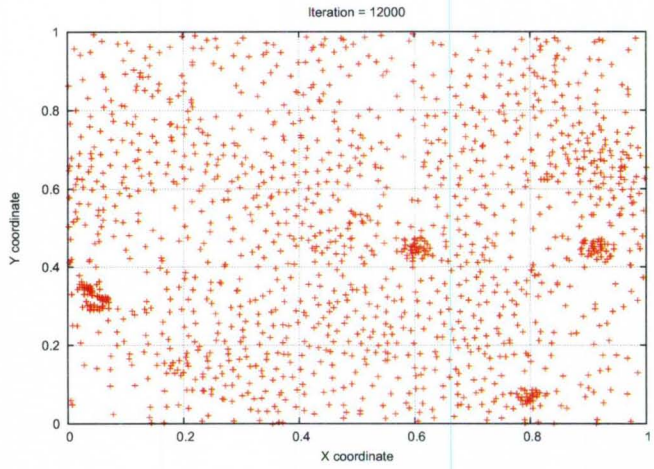
(b) Clustering result for all agents.

(c) Clustering result for dataset.

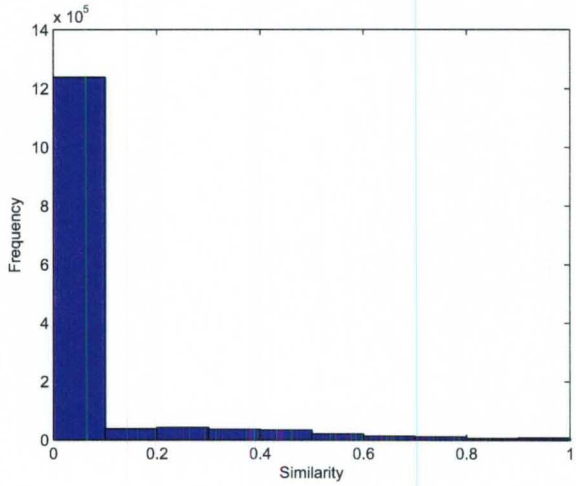
(d) Agent clusters generated after post processing and assigning agents to clusters.

Figure 11. Clustering a dataset with three clusters using FClust at iteration 30700.

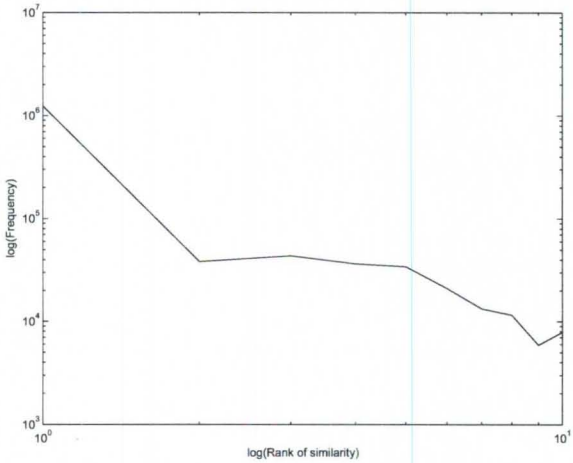
The next results are for two weeks worth of Web usage data for a Computer Engineering and Computer Science Department's website. We have chosen this data set, because it has have previously undergone extensive experiments and validation in [102, 101, 99], and hence it is considered a benchmark data set. In Figure 12(a), the algorithm did not converge because the similarity threshold computed according to Equation (24) was too high to form good clusters. With the average similarity and maximum similarity given in Table 9, the similarity threshold is computed as 0.53, which is very high given that the average similarity is 0.06. Therefore this example visually shows that the similarity threshold given in Equation (24) is not suitable for data with similarities distributed as a power law, as can be verified in Figure 12(b) and Figure 12(c) (the log-log plot exhibiting several linear segments).



(a) Agents for Web usage data,



(b) Cosine similarity histogram,

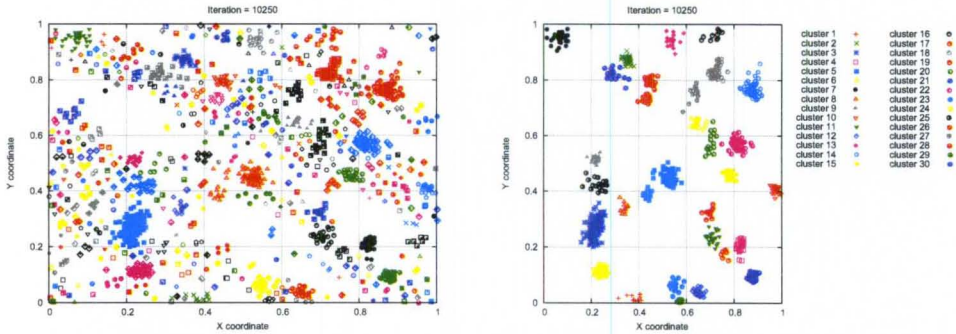


(c) Log-log plot of similarities.

Figure 12. Clustering the Web usage data using FClust with similarity threshold computed according to Equation (24). (a) no convergence because of an improper similarity threshold (b) Similarity histograms, (c) Log-log plot of similarities exhibiting power law properties.

TABLE 10
Several examples from the profiles of a Web usage data extracted using FClust where $S_{TH} = 10$ and $ICTF = 0.1$

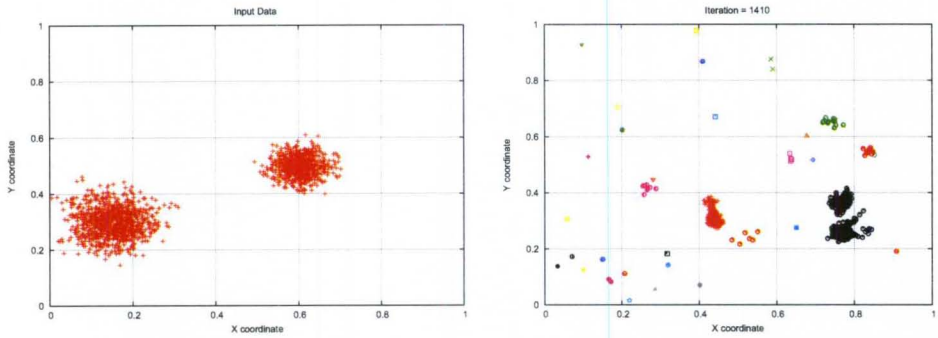
URL Frequency	URL
	Profile 1 (includes 211 sessions)
0.92	/
0.79	/cecs_computer.class
0.53	/courses.html
0.52	/courses_index.html
0.51	/courses100.html
0.27	/people.html
0.27	/people_index.html
0.27	/faculty.html
0.20	/courses300.html
0.19	/degrees.html
0.15	/courses200.html
0.13	/grad_people.html
0.12	/research.html
0.11	/courses_webpg.html
0.11	/index.html
0.10	/staff.htm
	Profile 2 (Includes 43 sessions)
0.93	/~joshi/courses/cecs352
0.33	/~joshi/courses/cecs352/slides-index.html
0.26	/~joshi/courses/cecs352/outline.html
0.23	/~joshi/courses/cecs352/text.html
0.23	/~joshi/courses/cecs352/handout.html
0.21	/~joshi/courses/cecs352/proj
0.19	/~joshi
0.16	/~joshi/courses/cecs352/proj/proj1.html
0.14	/~joshi/courses/cecs352/proj/overview.html
0.12	/~joshi/courses/cecs438
0.12	/~joshi/courses/cecs352/environment.htm



(a) Clustered agents before pruning.

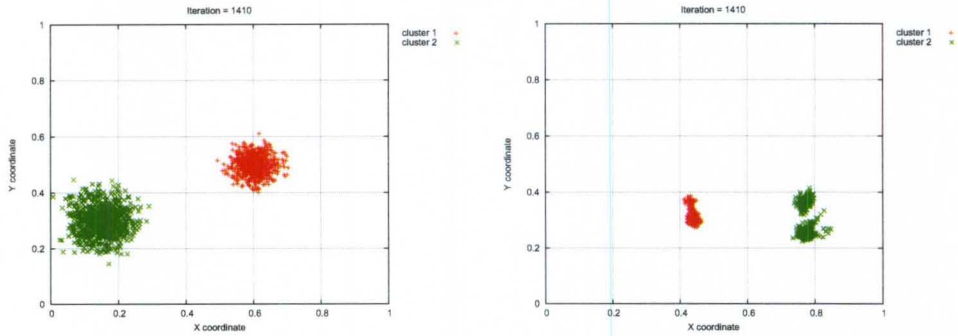
(b) After post-processing with Algorithm 13.

Figure 13. Generated profiles from web usage data using FClust in 10250 iterations.



(a) Dataset.

(b) Clustering result for all agents.



(c) Agent clusters generated after post processing and assigning agents to clusters.

(d) Agent clusters generated after post processing and assigning agents to clusters.

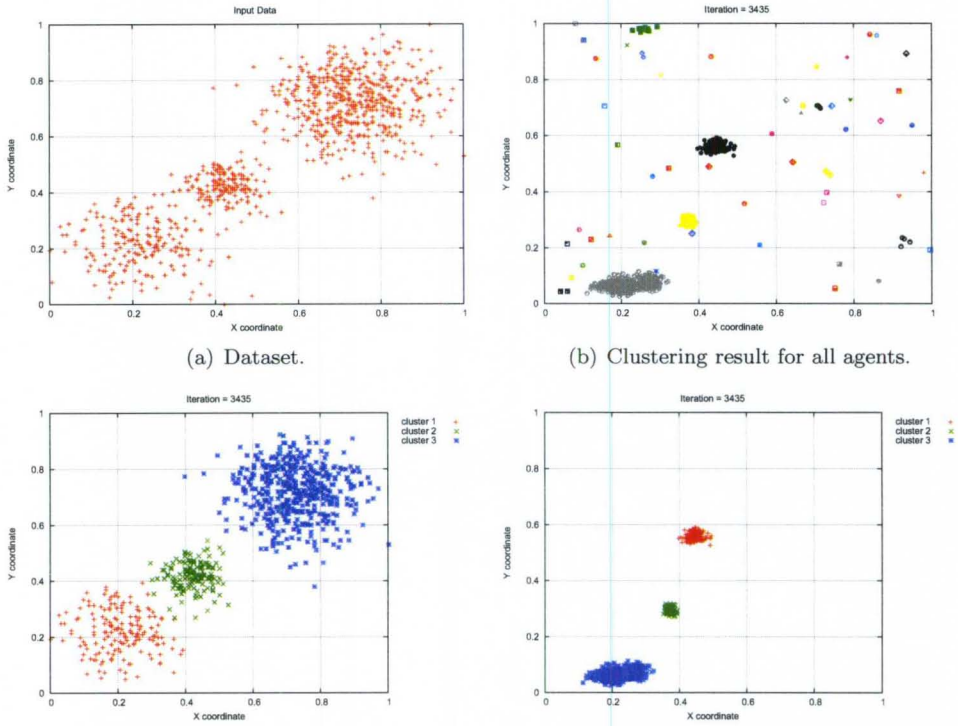
Figure 14. Clustering a dataset with two clusters using FClust-annealing where d_{th}=started from 1 down to 0.04, sim_{th}=0.91.

When we compute the similarity threshold using (26), it is 0.15. The value of α was set to 2.5 in this set of experiments. We obtain convergence as shown in Figure 13, where $d_{th} = 0.04$, $sim_{th} = 0.15$, $S_{TH} = 10$, and $ICTF = 0.10$. Several examples of the extracted profiles are presented in Table 10. For example, $Profile1$ represents a group of users (possibly prospective students) checking the department's main web pages. $Profile2$ represents a group of student users taking the course CECS 352 (*Joshi* is the instructor teaching the course).

FClust-Annealing Results for 2D and Web Usage Datasets

Figure 14 shows the clustering results for the dataset with 2 clusters using FClust-Annealing. Comparing this result with Figure 9, we can see that the annealing results in fewer iterations to convergence (1410 vs. 2792 iterations).

Figure 15 also shows that annealing not only accelerates the convergence, but also results in better quality clusters (no bridging effect). In Figure 10, even though there were 3 separate clusters in the agent space, after 24,400 iterations, as seen in Figures 10(b) and 10(d), clustering the data domain as shown in Figure 10(c) showed that the clustering process did not converge. Figure 11 also



(a) Dataset.

(b) Clustering result for all agents.

(c) Clustering result for dataset.

(d) Agent clusters generated after post processing and assigning agents to clusters.

Figure 15. Clustering a dataset with three clusters using FClust-annealing where d_{th}=started from 1 down to 0.04, sim_{th}=0.91, d_{ideal_th} for FClust and post-processing is 0.04.

confirms the fact that there is a tendency to combine three clusters into one cluster unless a better cluster formation algorithm is applied. Although the classical FClust suffers from these problems, FClust-Annealing clearly differentiates between these clusters. In addition to being more successful in clustering, FClust-Annealing converges in fewer iterations (3,435), compared to 24,400 iterations for FClust, i.e. it was 7 times faster.

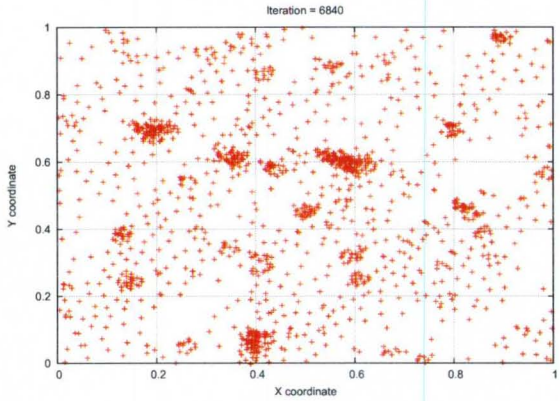
Figure 16 shows the results of clustering real life Web usage data, listed in Table 9, using FClust with annealing. Example profiles are shown in Table 11. Compared to Figure 13 which took 10,250 iterations, better quality clusters are now formed in only 6,840 iterations. Also 30 clusters were formed in Figure 13 whereas, with annealing, 25 clusters are formed. By checking the post-processed profiles, we observed that the decrease in the number is not a loss of information but rather a better convergence (broken clusters were combined). In FClust, some clusters were broken and their agents could not meet each other on the agents visualization panel and therefore could not be unified.

TABLE 11
Some samples from the Web user profiles, extracted using FClust-Annealing where $S_{TH} = 10$ and $ICTF = 0.1$

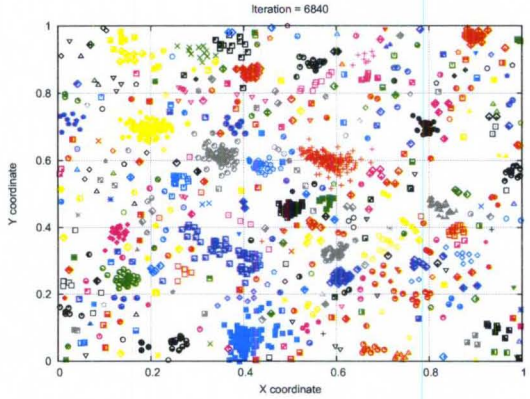
URL Frequency	URL
Profile 1 (includes 116 sessions)	
0.90	/
0.69	/cecs_computer.class
0.43	/courses_index.html
0.42	/courses100.html
0.41	/courses.html
0.29	/people.html
0.28	/people_index.html
0.28	/faculty.html
0.20	/courses300.html
0.18	/degrees.html
0.17	/courses200.html
0.13	/general.html
0.13	/general_index.html
0.13	/facts.html
0.13	/research.html
0.11	/grad_people.html
Profile 2 (Includes 31 sessions)	
0.90	/ joshi/courses/cecs352
0.35	/ joshi/courses/cecs352/slides-index.html
0.35	/ joshi/courses/cecs352/handout.html
0.35	/ joshi/courses/cecs352/outline.html
0.29	/ joshi/courses/cecs352/text.html
0.26	/ joshi/courses/cecs352/environment.html
0.13	/ joshi
0.13	/
0.13	/ joshi/courses/cecs438
0.13	/ joshi/courses/cecs352/proj

(K-means+FClust) Hybrid Results for 2D Datasets

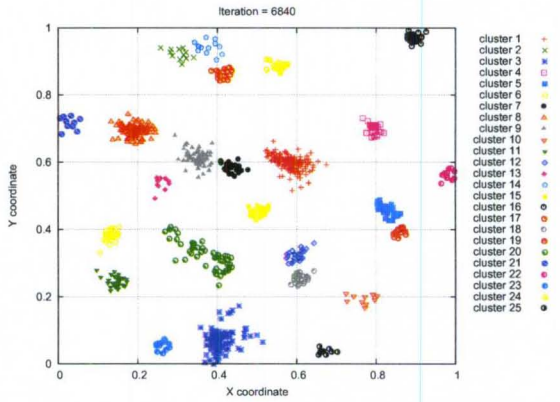
Figure 17 shows the results of the (K-means+FClust) Hybrid algorithm for Dataset I. The disadvantage of K-means is that it requires the number of clusters as input, however FClust can extract the number of clusters automatically. With the hybrid approach, 8 clusters are generated with K-means, as shown in Figure 17(b), where each agent was mapped to one cluster center generated by K-means. From these, FClust generated the 2 clusters, shown in Figure 17(c). Figure 17(b) represents the K-means result, (i.e. before starting the iterations of FClust). Compared to Figure 9, where 2,792 steps were needed for FClust's convergence, only 122 iterations were needed for the Hybrid version. Although the hybrid version speeds up the process considerably, it does not necessarily suffer from the bridging effect observed during the cluster extraction in post-processing.



(a) Clustering result for all agents.



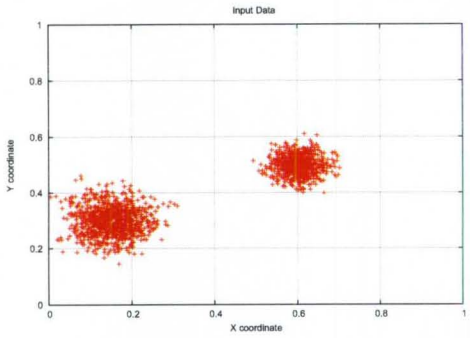
(b) Clustered agents before pruning.



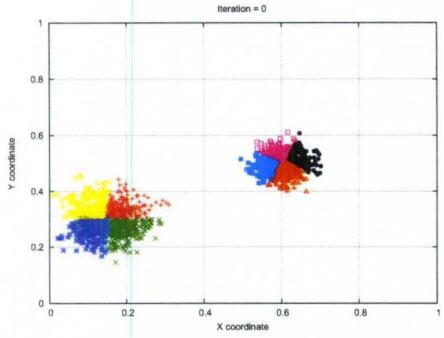
(c) Agent clusters generated after post-processing and assigning agents to clusters.

Figure 16. Clustering the Web usage session data using FClust-annealing where d_{th}=started from 1 down to 0.04, sim_{th}=0.15 , d_{ideal_th} for FClust and post-processing is 0.04.

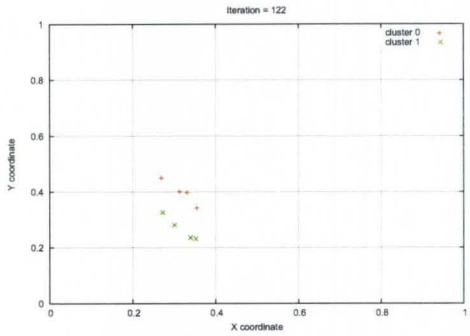
Figure 18 is a collection of figures showing the results of the (K-means+FClust) Hybrid Algorithm given in Algorithm 10 for Dataset II. When the results are compared with the simple FClust Algorithm in Figures 10 and 11, it can be observed that the hybrid algorithm is faster thanks to fewer iterations and to the modest linear computational cost.



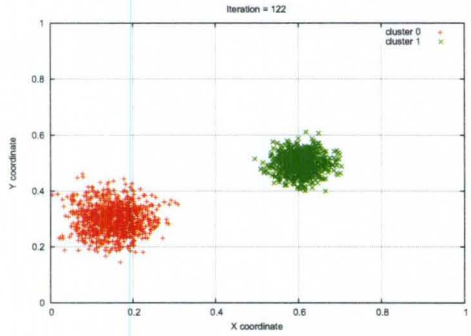
(a) Dataset.



(b) Data clustered into $K = 8$ clusters using K-means.



(c) Agent clusters after applying thresholding.

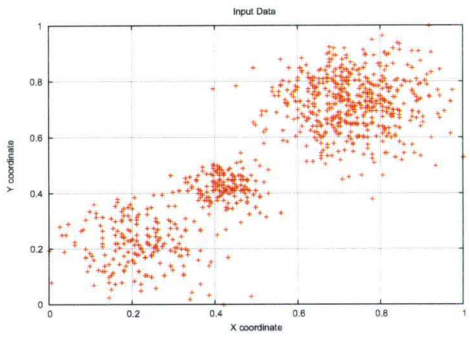


(d) Data clusters after applying thresholding.

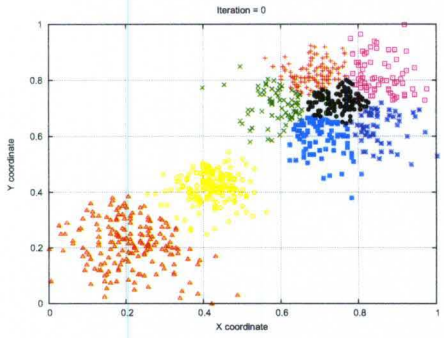
Figure 17. Stable output of (K-means+FClust) hybrid on a 2 cluster-data set where sim_{th}=0.88 (using Eqn.(24)), d_{th}= 0.4. d_{ideal_th} for FClust=0.04, d_{ideal_th} for forming clusters = 0.08.

FClust, FClust-Annealing and (K-means+FClust) Hybrid Results for Iris and Pima Datasets

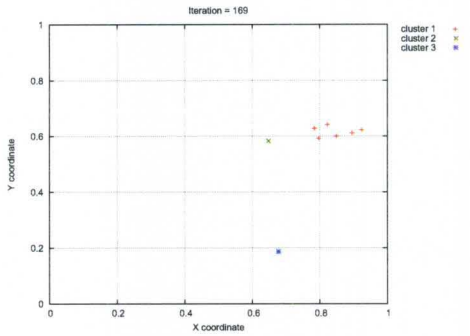
Table 12 compares the FClust, FClust Annealing, and (K-Means+FClust) Hybrid algorithms on the datasets Iris and Pima, which were also used in [117]. The results were averaged over 10 different runs, and the numbers inside the parentheses represent the standard deviations. We have also observed, in the Iris dataset experiments, that the original FClust algorithm tends to get stuck in local optimum for a long time, but after enough iterations, it can find a better clustering. For example, in one run, FClust produced 4 clusters around 1500 iterations, with a cluster error rate of approximately 0.25, computed via Algorithm 3, while around 2500 iterations, it formed 3 clusters. However, the error rate was still high (0.27). Later, the number of clusters dropped to 2 and around 7000 iterations, it found 3 clusters with an error rate of 0.08. More iterations resulted in high error rates (around 0.22) again with 2 clusters. So the system will cycle between 2, 3 and 4 clusters. These are different but somehow stable clustering options. However, it was observed that, results with more than 5 clusters with the given parameters, are a result of an insufficient number of iterations. Again



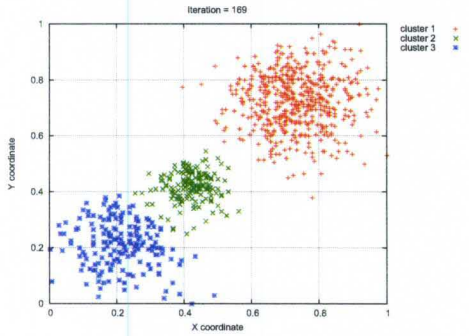
(a) Dataset.



(b) Data clustered into K=8 clusters using K-means.



(c) Agent clusters after applying thresholding.



(d) Data clusters after applying thresholding.

Figure 18. Stable output of (K-means+FClust) hybrid on a 3 cluster-data set where sim_{th}=0.73 (using Eqn.(24)), d_{th}= 1.0. d_{ideal_th} for FClust=0.04, d_{ideal_th} for cluster forming =0.08.

as we have noted in Section 3.2.1, since every pairwise ideal distance is one constraint/objective to be satisfied, there are $n \times (n-1)/2$ objectives. Theoretically this can lead to up to $n \times (n-1)/2$ Pareto solutions on the Pareto front. The actual number in practice is much less however, since several of these constraints can be satisfied simultaneously.

The FClust-Annealing version decreases the number of iterations drastically. It starts forming cluster centers from the first iterations, and even though the centers are created in less than 10 or 20 steps, they still seem to be reasonably good. For the first few iterations, the cluster centers changed rapidly since the neighborhoods were wide. Yet clusters were still formed after post-processing. Finally, the clustering scheme which gave the minimum error was reported. Annealing converges to a meaningful cluster formation faster than FClust. Moreover, in just a few iterations, it can already present different possible clustering options. This process and the changes in the formed cluster numbers and their errors with the iterations, for 10 different runs on the Iris dataset, are shown in Table 13. For each run, the first row is the number of clusters generated at the corresponding iteration number, which is given as the column label. Similarly, the second row shows the error computed via Algorithm 3.

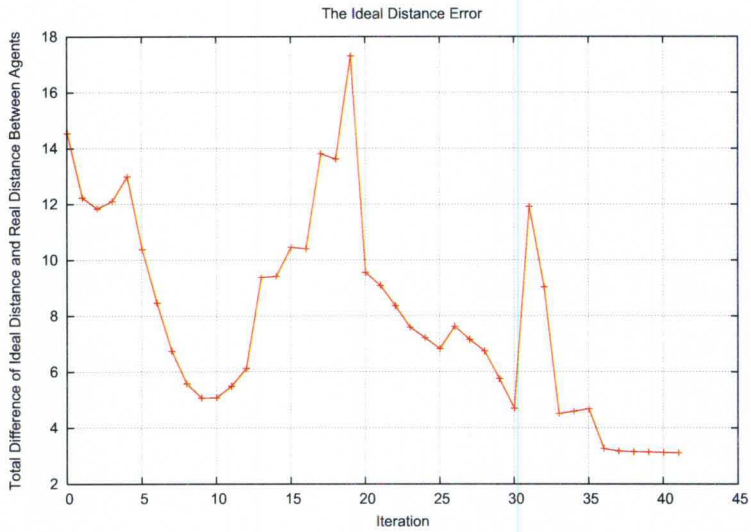


Figure 19. Evolution of the ideal distance error with iterations for clustering the Iris dataset using (K-means+FClust) Hybrid algorithm.

In FClust-Annealing and the (K-means+FClust) Hybrid, since the neighborhood is wider, clusters are formed faster, and there will be fewer agents on the visualization panel between agent flocks. Later (in the case of annealing), when the neighborhoods become narrow, the chance of flocks meeting and affecting each other decreases, which causes a decrease in the exploration for better clustering options. Therefore FClust Annealing and the (K-means+FClust) Hybrid are more prone to getting stuck in local optima. Some random moves could be added to the algorithm to increase the opportunity for exploration. One of the problems observed with the (K-means+FClust) Hybrid on the Iris data set was that the algorithm may separate the members of the first cluster into two groups. During the clustering using the Hybrid algorithm, K was set to 8 in K-means, d_{ideal_th} was 0.04, and during post processing, it was 0.08.

To stop the Hybrid algorithm, the *ideal distance error*, for each pair of agents is computed, and when its difference compared to the previous iteration fell below 0.009, the algorithm was stopped. Figure 19 shows the ideal distance error versus the iteration number for the Iris dataset using the (K-means+FClust) Hybrid algorithm. The irregularities observed as sudden increases in the error correspond to the time when clusters reached the border of the visualization panel and continued moving, thus wrapping around toward the opposite side of the panel. That said, the (K-means+FClust) Hybrid algorithm produced reasonable clustering results in fewer iterations on the Iris dataset.

In the experiments with the Pima data, FClust formed 2 or 3 clusters, for d_{th}=0.04 with

an error rate around 0.50. Manual stopping terminated earlier than automated stopping because, a Human typically follows the bigger flocks of agents, thus the agents which are spread around the visualization panel do not affect the human observer as much as they may affect the automated stopping. In the FClust-Annealing algorithm, 2 clusters of the Pima dataset were formed around 20 to 30 iterations, with a cluster error rate between 0.45 and 0.50. For the Pima data set, with smaller cluster size threshold values, clusters would be observed in fewer iterations. However, we kept this threshold constant as $n/20$ to be able to compare the proposed algorithms with the original FClust algorithm. Table 14 shows that an average of 1651 iterations for FClust-annealing were sufficient to get a state of the visualization panel which would require an average of 3995 iterations of the original FClust algorithm. Similarly, K was 8 and d_{ideal_th} was 0.04 for the the Hybrid algorithm. During post processing, d_{ideal_th} was 0.08. We also observed that, for both the Iris and Pima data sets, minimum cluster errors were observed for the FClust-Annealing experiments (0.11 for Iris and 0.43 for Pima) compared to all other algorithms.

(SPKM+FClust) Hybrid Results for the Web Usage Dataset

Using the Web usage data set, the (SPKM+FClust) hybrid algorithm was tested and compared to FClust. In both algorithms, the ideal distance was computed using (28). Our experiments showed that since Web usage data followed a power low distribution [129], (24) resulted in a grossly over-estimated similarity threshold and led to no cluster formation, as mentioned in Section 2.5.2.

Although FClust managed to converge thanks to the similarity threshold modification, and although meaningful profiles were extracted, we observed that the results contained too many invalid clusters (i.e. clusters with less than S_{TH} sessions), as shown in Table 15. In fact, on the average, after 4000 iterations, only 31 out of 502 clusters were valid. Note that the notations *Intra Sim* and *Inter Sim* stand for intra- and inter-cluster similarities, which ideally, should be highest and lowest, respectively. *Gen. Clust.* is the number of generated clusters, before pruning the clusters with fewer than S_{TH} sessions. *Final Clust.* represents the final number of clusters after pruning. The values in parenthesis are the standard deviations over 10 runs.

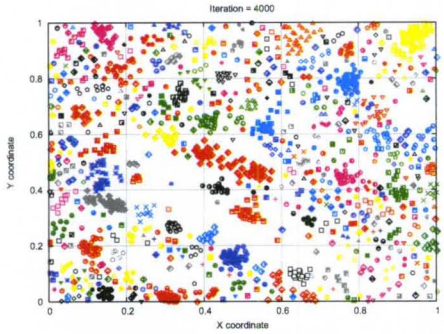
Table 16 lists the evaluation metrics for the (SPKM+FClust) Hybrid, averaged over 10 different runs, and compares results for different K values used for the number of clusters in SPKM. As expected, when the number of agents decreases, the number of iterations needed to converge also decreases. Selecting the number of agents, K, not only affects the number of iterations but also affects the quality of the resulting clusters. Increasing K, increases the number of (agent-space

distance to ideal distance matching) constraints that need to be satisfied. However, a higher value of K is also crucial to give a chance for collaboration to actually take place, and thus extract higher quality clusters. The presence of more agents leads to stronger collaboration, whereas a sparse agent space is very risky because it reduces the potential for collaboration, the very mechanism by which this *social* learning paradigm actually works.

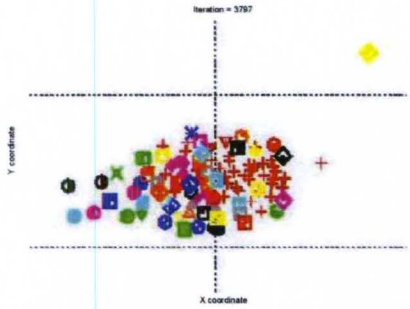
Figure 20 compares the clustering results of FClust and our proposed Hybrid approach. Even though a very small number of agents can degrade the collaborative power of the algorithm, having a smaller number of agents on the visualization panel makes it easier to observe the similarity relationships between the data records. Figures 20(a) and 20(b) show that while the hybrid algorithm has converged around 4000 iterations, FClust needed more iterations to cluster the unlabeled agents. Table 18 compares the average system CPU time computed by 10 different runs of the tested algorithms, showing a big advantage for the hybrid approach mainly because it does not have to compute a complete distance/similarity matrix for the entire data, but rather only for the reduced cluster centroids resulting from SPKM. The dominant part of the computation time is the computation of ideal distances amongst agent pairs, which is performed only once at the start. Thus, the difference in computation times between $4,000$ and $10,000$ is small This is also expected from the complexity analysis in Sec. 2.5.2. Furthermore, Tables 15 and 16 show that the quality of the discovered clusters is also higher in the proposed hybrid algorithm, since the average similarity within clusters is almost two times higher (thus better), and the average inter-cluster similarity is almost 5 times lower (thus better). During our experiments, we observed that even at a whooping $40,000$ iterations, FClust still could not reach the quality levels of the proposed hybrid on the evaluation metric. A few sample profiles from Figure 20(d) are listed in Table 17 (paucity of space prevented us from listing all 19 profiles), showing results that fare very well with the results in [98] on this same benchmark data set: A significant number of good profiles were discovered, and the profiles are interpreted as meaningful groups (thus patterns) of user activities on the website that range from casual visitors of the website to students who are more focused on particular courses.

3.4 Conclusions

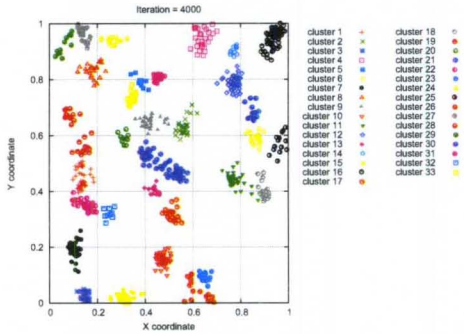
Flocks-of-agents based swarm intelligence holds a great promise, and being the most recent of all the swarm-based clustering approaches, it has been the least studied. To address some of the limitations of the standard flock-based clustering algorithm (FClust) on Web usage data, we developed some improvements, including the K-means variant Hybrid algorithm. Our hybrid ap-



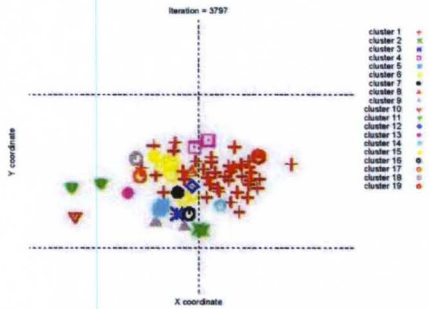
(a) Clustered agents beforepruning (FClust).



(b) Clustered agents beforepruning (Hybrid).



(c) Agent clusters generated after post-processing and assigning agents to clusters (FClust).



(d) Agent clusters generated after post-processing and assigning agents to clusters (Hybrid).

Figure 20. Clustering the Web usage session data using FClust vs. (SPKM+FClust) Hybrid where $d_{th} = 0.4$, $d_{ideal\_th} = 0.04$.

proach reduces the quadratic complexity of FClust to linear complexity and performs similarly to FClust with fewer iterations for clustering high-dimensional data such as web usage data and text documents. Our experiments confirmed the superiority of the proposed hybrid approach, both in terms of quality of the final results and computational cost.

TABLE 12. Compared Results (Averaged over 10 runs).

Dataset	Number of Items	Number of Attributes	Number of Classes	FClust			FClust Annealing			FClust Hybrid		
				Average Clusters Found	Avg. Error	Avg. Iter. No	Average Clusters Found	Avg. Error	Avg. Iter. No	Average Clusters Found	Avg. Error	Avg. Iter. No
Iris	150	4	3	3.1 (0.32)	0.18 (0.04)	1811 (400.1)	2.9 (0.32)	0.16 (0.03)	11.6 (4.33)	4.5 (0.85)	0.22 (0.06)	41.5 (22.45)
Pima	768	8	2	2 (0)	0.47 (0.01)	3995.1 (984.6)	2 (0)	0.475 (0.02)	19.9 (4.95)	3.8 (1.5)	0.45 (0.03)	41.7 (18.58)

TABLE 13. Number of clusters extracted and corresponding error at different iteration steps for 10 different runs of clustering the Iris data using FClust-Annealing where d_{th}=0.4, d_{ideal_th}=0.04

Iteration No	1	10	50	100	200	300	400	500	1000	1500	2000	2500
Run 1	1	3	6	6	5	6	5	4	4	4	3	4
	0.67	0.16	0.26	0.26	0.24	0.24	0.18	0.16	0.15	0.15	0.13	0.15
Run 2	0	3	2	2	2	2	2	2	2	2	3	2
		0.15	0.23	0.23	0.23	0.23	0.23	0.22	0.23	0.24	0.22	0.22
Run 3	0	1	5	5	7	7	5	4	4	4	2	3
		0.67	0.23	0.21	0.24	0.24	0.21	0.19	0.19	0.19	0.22	0.18
Run 4	0	2	2	2	2	2	2	3	3	4	4	3
		0.24	0.23	0.22	0.22	0.23	0.22	0.17	0.17	0.18	0.19	0.09
Run 5	0	2	4	4	4	4	4	4	2	3	3	4
		0.32	0.23	0.27	0.27	0.28	0.28	0.28	0.23	0.18	0.18	0.11
Run 6	1	0	4	4	3	3	4	3	5	5	4	4
	0.67		0.28	0.2	0.17	0.18	0.27	0.22	0.21	0.22	0.18	0.19
Run 7	0	1	3	5	4	4	3	3	3	4	4	4
		0.67	0.19	0.16	0.14	0.17	0.15	0.16	0.15	0.1	0.13	0.11
Run 8	0	0	5	6	4	5	4	4	4	2	2	2
			0.26	0.28	0.22	0.2	0.22	0.22	0.26	0.22	0.22	0.22
Run 9	1	0	5	3	3	3	3	3	2	2	2	2
	0.67		0.19	0.13	0.15	0.13	0.14	0.14	0.22	0.22	0.22	0.22
Run 10	0	1	4	4	4	6	4	3	3	2	2	3
		0.67	0.27	0.22	0.24	0.22	0.13	0.16	0.17	0.22	0.23	0.1
Avg. No of Clusters	0.3	1.3	4	4.1	3.8	4.2	3.6	3.3	3.2	3.2	2.9	3.1
Average Error	0.67	0.411	0.237	0.218	0.212	0.212	0.203	0.192	0.198	0.192	0.192	0.159

TABLE 14
Average result of 10 different runs of clustering the Pima data set using FClust-Annealing, where d_{th} started from 1 and decreased to 0.04.

	Found Clusters	**Error**	**Iteration No**
Average	2	0.47	1651.1
(Standard Deviation)		(0.01)	(587.3)

TABLE 15
Average results (and standard deviations in brackets) of 10 different runs of FClust algorithm, where $d_{th} = 0.04$, $\alpha = 2.5$, $S_{TH} = 10$, $ICTF = 0.10$.

FClust Before Post Processing				**FClust After Post Processing**				
Intra Sim	**Inter Sim**	**Inter/ Intra**	**Gen. Clust.**	**Intra Sim**	**Inter Sim**	**Inter/ Intra**	**Final Clust.**	**Iter.**
0.891	0.014	0.016	501.8	0.352	0.108	0.307	31.4	4000
(0.004)	(6E-04)	(8E-04)	(12.5)	(0.013)	(0.015)	(0.048)	(1.96)	(0)

TABLE 16. Average result (and standard deviations in brackets) of 10 different runs of the (SPKM-FClust) hybrid algorithm with different K values, where $d_{th} = 0.4$, $d_{ideal\_th} = 0.04$, $\alpha = 2.5$, $S_{TH} = 10$, $ICTF = 0.10$.

SPKM				**FClust Hybrid Before Post-Processing**				**FClust Hybrid After Post-Processing**				
Intra Sim	**Inter Sim**	**Inter/ Intra**	**K**	**Intra Sim**	**Inter Sim**	**Inter/ Intra**	**Gen. Clust.**	**Intra Sim**	**Inter Sim**	**Inter/ Intra**	**Final Clust.**	**Iter.**
0.537 (0.019)	0.029 (0.003)	0.0538 (0.0044)	100 (0)	0.428 (0.034)	0.007 (0.002)	0.016 (0.005)	54.5 (6.6)	0.572 (0.018)	0.021 (0.007)	0.037 (0.012)	25 (5.6)	3870.8 (2410.1)
0.511 (0.031)	0.032 (0.007)	0.0621 (0.0145)	50 (0)	0.441 (0.045)	0.013 (0.005)	0.029 (0.01)	30.4 (5.04)	0.498 (0.032)	0.02 (0.005)	0.04 (0.008)	22.9 (5.02)	934.2 (600.9)
0.496 (0.032)	0.033 (0.007)	0.0679 (0.0163)	30 (0)	0.463 (0.054)	0.021 (0.008)	0.045 (0.018)	23 (2.16)	0.452 (0.039)	0.025 (0.008)	0.056 (0.019)	19.8 (2.97)	112.8 (127.82)

TABLE 17
8 sample profiles out of a total of 19 Web user profiles visualized in Figure 20(d), and extracted using the (SPKM+FClust) Hybrid with $K = 100$, $S_{TH} = 10$ and $ICTF = 0.1$.

URL Freq.	URL	URL Freq.	URL
Profile 1		**Profile 11**	
0.47	/	0.72	/ shi/cecs345
0.37	/cecs_computer.class	0.50	/ shi/cecs345/Homeworks/1.html
0.23	/courses_index.html	0.42	/ shi/cecs345/Projects/1.html
Profile 2		**Profile 7**	
1.00	/ yshang/CECS341.html	0.97	/access
0.60	/ yshang/W98CECS341	0.85	/access/details.html
Profile 6		**Profile 13**	
0.82	/ saab/cecs333/private	0.94	/degrees_grad.html
0.70	/ saab/cecs333	0.88	/degrees_grad_index.html
0.48	/ saab/cecs333/private/assignments	0.84	/deg_grad_genor.html
0.41	/ saab/cecs333/private/lecture_program	0.64	/degrees.html
Profile 10		**Profile 19**	
0.97	/ c697168/cecs227/labs/main.html	0.98	/ saab/cecs303/private
0.97	/ c697168/cecs227/labs/lab1.html	0.90	/ saab/cecs303
0.96	/ c697168/cecs227/labs/left.html	0.83	/ saab/cecs303/private/solution

TABLE 18
Comparison of average system CPU times of 10 different runs for FClust (4000 and 10000 iterations) and (SPKM-FClust) hybrid algorithm with 4000 iterations, where $K = 100$, $d_{th} = 0.4$ in the Hybrid and 0.04 in FClust , $d_{ideal\_th} = 0.04$, $\alpha = 2.5$, $S_{TH} = 10$, $ICTF = 0.10$.

	FClust-4000	FClust-10000	Hybrid
Avg. CPU Time (s)	6.98	7.28	0.04
(Standard Deviation)	(0.1)	(0.2)	(0.02)

CHAPTER 4

FLOCKS OF AGENTS BASED RECOMMENDER SYSTEM

In this chapter, we define a new Swarm Intelligence-based recommender system (FlockRecom) based on the collaborative behavior of bird flocks for generating Top-N recommendations. The flock-based recommender algorithm (FlockRecom) iteratively adjusts the position and speed of dynamic flocks of agents on a visualization panel. By using the neighboring agents on the visualization panel, top-n recommendations are generated. The performance of FlockRecom is evaluated using the Jester Dataset-2 [2] and is compared with a traditional collaborative filtering based recommender system. Experiments on real data illustrate the workings of the recommender system and its advantages over its CF baseline.

4.1 Introduction

In recent years, we have witnessed an explosive growth in the amount of information. Each day, more books and journals are published, more newspaper articles are written, more web pages are posted online, more office documents are prepared, more photos are taken, and more movies are created. This over-abundance of information contributes to the reasons why we can get hundreds or even thousands of results for a simple search, and why we can find it hard to arrive at the resources that we need by wading through endless labyrinths of Web pages and Websites. This problem is commonly referred to as *information overload.*

Recommender systems aim to assist users in handling the information overload problem. Two of the many approaches to build recommender systems include collaborative filtering (CF) and swarm intelligence (SI), both built on the collaboration of users. Based on the assumption that users with similar past behaviors (rating, browsing, or purchase history) have similar interests, a collaborative filtering system recommends items that are liked by other users with similar interests [132, 133, 69]. More information on CF and other approaches are presented in Section 2.8. In this chapter, we present a new recommender system approach using a Swarm Intelligence algorithm, inspired from bird flocks and called flocks-of-agents based recommender system (FlockRecom). In

this approach, each user is mapped to one agent, i.e. each agent of the flock represents a user. Initially, agents are placed on a planar surface (hereinafter referred to as the *visualization panel*). Then, in each iteration, similar agents attract each other, while dissimilar agents repel each other. Thus, the agents' speed gets updated according to their neighboring agents. In time, similar agents start moving together and closer, forming clusters [129, 128, 117]. Moreover, the distance between the agents depends on the similarity between the users that are mapped to those agents. At each iteration, recommendations are generated/updated using the neighboring agents. Agents keep moving until they are forced to stop. Thus, the dynamic character of the FlockRecom provides dynamic recommendations, making FlockRecom stronger at exploring different recommendation options and providing more variety for recommendations. Variety or diversity is important in the environments that users keep visiting repeatedly. Additionally, initial experimental results show that, FlockRecom is a promising approach for recommendation in dynamic environments, thus holding a potential for social networking platforms.

In the rest of this chapter, we present FlockRecom in Section 4.2 and experiments on a real life dataset in Section 4.3. Finally, we make our conclusions and discuss future work in Section 4.4.

4.2 Flocks-of-Agents based Recommender System

The inspiration behind the flocks-of-agents-based recommender system stems directly from the collaboration among bird flocks in nature. Flocks of agents-based recommender system is suitable for any kind of data set where one can define a similarity measure between users. Each agent represents one user. Initially, agents are placed on a *visualization panel*, which is a 2 or 3-dimensional continuous space, where x, y (and if applicable z) coordinate values range between 0 and 1. Agents may be placed randomly or some background information can be used to place them. Then, they start moving around. As they meet other agents in a defined neighborhood, they try to remain within an ideal distance to each other, which is determined according to the similarity of the users that agents are representing. The more the users are similar, the smaller the ideal distance will be. Ideal distances are computed for each agent pair once at the beginning of the algorithm based on the intrinsic properties or ratings of the users. If neighboring agents are further apart than the ideal distance, there will be an attraction force between them and the agents will try to move closer to each other. In contrast, if the distance is less than the ideal distance, then there will be a rejection force, and agents will move apart from each other. Given this basic idea, Algorithm 16 gives the procedure for *Flocks of Agents Recommender System (FlockRecom)*.

Algorithm 16 FlockRecom Algorithm

Input: Dataset.

Output: Top-N recommendations.

1: Initially place the agents on the visualization panel
2: Initialize velocities of all agents
3: Compute the ideal distances, d_{ideal}, between agents.
4: **while** 1 **do**
5: **for** each agent i **do**
6: **for all** j such that $d(j,i) \leq d_{th}$ and $i \neq j$ **do**
7: **if** $d(i,j) = d_{ideal}(i,j)$ **then**
8: $\beta(i,j) \leftarrow 0$
9: **else if** $d(i,j) > d_{ideal}(i,j)$ **then** // attraction
10: $\beta(i,j) \leftarrow 4 \times \left(\frac{d(i,j)-d_{ideal}(i,j)}{d_{th}-d_{ideal}(i,j)}\right)^2$
11: **else** // repulsion
12: $\beta(i,j) \leftarrow -4 \times \left(1 - \frac{d(i,j))}{d_{ideal}(i,j)}\right)^2$
13: **end if**
14: $\overline{v}_{resulting}(i,j) \leftarrow \overline{v}(i) + \overline{v}(j) + \beta(i,j) \times \overline{v}_{cap}(i,j)$
15: **end for**
16: **if** $\exists\ j$ such that $d(j,i) \leq d_{th}$ and $i \neq j$ **then**
17: $\overline{w}(i)$ = normalize ($\sum_{j|d(j,i)\leq d_{th}\&i\neq j} \overline{v}_{resulting}(i,j)$)
18: **if** The angle between $\overline{v}(i)$ and $\overline{w}(i)$ is less than or equal to 90 degrees **then**
19: $\overline{v}_{next}(i) \leftarrow \overline{w}(i)$
20: **else**
21: $\overline{v}_{next}(i) \leftarrow \overline{v}(i)$
22: **end if**
23: **else**
24: $\overline{v}_{next}(i) \leftarrow \overline{v}(i)$
25: **end if**
26: amp_{next}(i) $\leftarrow amp_{def} + \frac{\left|\sum_{j|d(j,i)\leq d_{th}\&i\neq j} \overline{v}_{resulting}(i,j)\right|}{100}$
27: **end for**
28: **for** each agent i **do**
29: compute new position $p_{next}(i) \leftarrow p_{current}(i) + amp_{next}(i) \times \overline{v}_{next}(i)$
30: **end for**
31: Move all agents to the updated positions and update current velocities.
32: **for** Each user u that will be provided recommendations **do**
33: Let i_u be the agent representing u
34: Let $S(u) = \{$users corresponding to neighboring agents of i_u within distance d_{ideal_th}
35: Compute average rating per item among all users in $S(u)$
36: Select top-N items as N items with highest average rating
37: Recommend Top-N items to user u
38: **end for**
39: **end while**

In steps 1 and 2, the initialization is performed. The velocity vector $\overline{v}$, is a unit vector, (i.e. $||\overline{v}|| = 1$), representing the direction. In step 3, the ideal distances between agents are computed via Equation (27), where $sim(i,j)$ is the similarity between the users that agents i and j are representing,

and sim_{th} is the similarity threshold. We use cosine similarity to compute the similarity between two users, as shown in Equation (6).

Later, for each agent i, the neighboring agents that are close enough to i on the visualization panel, are extracted in Line 6, where *d(i,j)* is the 2D Euclidean distance between agents i and j, and d_{th} is the distance threshold. Then, for each neighbor:

- If the distance between the agents i and j is equal to the ideal distance between them (Line 7), there is no attempt to change i's velocity due to j (Line 8).
- If the distance between the agents i and j is greater than the ideal distance between them (Line 9), an attraction force will move i closer to j, with a more similar velocity to j (Line 10).
- If the distance between the agents i and j is smaller than the ideal distance between them (Line 11), a repelling force will move i further from j, with a less similar velocity to j (Line 12).

In line 14, the velocity effect on i due to neighbor j is computed where $\overline{v}_{cap}(i,j)$ is the unit vector pointing from i to j. Next is the computation of the updated velocity of agent i, $\overline{v}_{next}(i)$, between lines 16 and 25. First, if i has neighbors, then their resulting velocities on i are summed up and normalized. If the total, normalized velocity $\overline{w}$, does not change the agent's current direction more than 90 degrees, then the updated velocity is assigned as $\overline{w}$. Otherwise the velocity is kept unchanged for the next iteration. Similarly, if agent i does not have any neighbors -note that an agent is not considered to be a neighbor of itself- then the velocity will be kept the same for the next iteration. In line 26, the amplitude is computed depending on the number of neighbors and distance threshold, where amp_{def} is the default minimum amplitude. The minimum amplitude is empirically set to 0.02. At the end of each iteration, the updated agent coordinates are computed, and all the agents are moved to their updated positions simultaneously (lines 28 to 31). Then the next step is to generate recommendations. Let's call *active users* to the users that recommendations are generated for. For each active user, a set of users are determined via the agent neighborhood on the visualization screen. In other words, for the active user u's corresponding agent i_u, neighbor agents set $S(j)$ is determined such that $d(i,j) \leq d_{ideal\_th}$ and $i \neq j$. Then, average item ratings are computed for all users represented by the agents in $S(j)$. Finally, n items with the highest average ratings are recommended to user u. These n items are called *top-n* items.

4.3 Experiments

The experiments were conducted on a dataset extracted from the Jester Dataset-2, which is available online on the website of the University of California, Berkeley [2]. In addition to FlockRecom, a traditional collaborative filtering based recommender system was also implemented to test the performance of our system, and performances were evaluated and compared using precision, recall, and F1 metrics as well as variety in the recommendations.

4.3.1 Dataset and Pre-Processing

TABLE 19
Dataset.

Dataset ID	Number of Users	Number of Items	Avg. Sim.
Jester	50	150	0.29
Jester II	500	150	0.3

In our experiments, we used the Jester Dataset-2, which is a collection of user ratings for 150 different jokes [2]. The dataset has 63,978 users, and the ratings range on a real value scale from -10 to +10 (-10 and +10 are included). As shown in Table 19, in the experiments, the first 50 or 500 users were used with all 150 jokes, thus 1911 or 19771 ratings were used. The user ratings for jokes were in the scale of -10 to +10. In the pre-processing phase these ratings were normalized in the scale 0 to 1, where 0 indicates that the item is rated -10 or is not rated by the corresponding user [18].

4.3.2 Evaluation Metrics

Evaluating a recommender system can be nearly as hard as designing and implementing the system, in part because no simple, objective, and general agreed upon mathematical formula is always available to measure success [132, 12, 126, 20, 19]. One problem suffered by some systems is *over-specialization*. When the recommendations are limited to the user's behavior, user's profile, or user's ratings, the user can be restricted to seeing only similar items, and there will be no randomness. In artificial intelligence, this problem is known as, the exploration/exploitation dilemma.

Evaluating information retrieval systems can be done if one has available, a set of user queries and a labeled set of search results (relevant and non-relevant). In this case, *precision* (proportion

of retrieved items that are really relevant) and *recall/coverage* (proportion of all items, known to be relevant, that are retrieved) are typically used as goodness metrics [126, 12]. One method for evaluating a recommender system relies on asking for a ranking or a rating of the results from the users. However, this can be subjective. Moreover, if the study is for research purposes, it can be hard to find a sufficient number of real users with diverse interests for the experiment. For this reason, historical data has also been used in research studies. In this case, the output of a recommender system is compared to the real interests of the user in the historical data, and metrics such as precision and coverage are computed [126, 12]. One popular way to assess the success of a system is to compare it, for example with recommending the default, most popular, or even a randomly selected item.

In our experiments, we test how well the recommendations can represent the user while providing a variety of recommendations to address the exploration-exploitation dilemma. Thus, recommendations should be as close as possible to the real interests of the user. In our experiments, the ground truth is the set of items liked by the active user. An item is considered liked by the user if its rating exceeds 0.7, after normalization to a scale of 0 to 1, where 0.7 was chosen based on existing literature [57].

Evaluation should take into account both (i) precision (the recommended items are all correct or included in the ground truth set of items)

$$precision = \frac{|I_R \cap I_G|}{|I_R|} \tag{33}$$

and (ii) coverage/recall (the recommended items are complete compared to the ground truth set, i.e. they include all the ground truth items).

$$recall = \frac{|I_R \cap I_G|}{|I_G|} \tag{34}$$

where $I_G = \{\text{Items liked by user}\}$ represent the ground truth set of items and $I_R = \{\text{Items recommended by system}\}$ represent the set of recommended items.

A precision score of 1 indicates that every recommended item was part of the ground truth set, whereas, a recall score of 1 represents that all the ground truth items were recommended. Precision and recall can be combined in the F1 measure, given in Equation (35)[57]. Higher values of the F1 measure indicate a more balanced combination of high precision and recall.

$$F1 = 2 \times \frac{precision \times recall}{precision + recall} \tag{35}$$

In addition to precision, recall, and F1, we compute variety or diversity by counting the number of unique items recommended to the user.

Definition 1 *Variety = The number of distinct recommended items* $= |I_R|$

4.3.3 Experimental Results

In the experiments, using FlockRecom, we generated top-n recommendations for active users at each iteration of our dynamic FlockRecom algorithm, and we varied n from 1 to 30. In our previous studies on using FClust for clustering, having a distance threshold small as 0.04 was superior for many other datasets. However, by trial and error we observed that having the distance threshold d_{th} as high as 0.4 produced better results.

To compare the results, a traditional user-based nearest neighbor CF algorithm [132] was used. In the collaborative filtering approach, the neighbors of an active user u are defined as the users that are similar to u above a similarity threshold sim_{th} as given in Equation (36). After the neighbors are computed, the average item ratings per active user are evaluated using Equation (37), where u is the active user and i is an item.

$$CF\_neighbor(u) = \{t | similarity(u,t) \geq sim_{th}, t \neq u\} \tag{36}$$

$$Average\_rating(u,i) = \frac{\sum_{t \in \text{CF\_neighbor(u)}} rating(t,i)}{|CF\_neighbor(u)|} \tag{37}$$

In the experiments, the cosine similarity was used to compute the similarity between users. After trial and error, the similarity threshold sim_{th} was set to 0.07, whereas the distance threshold d_{th} was set to 0.4. The evaluation metrics were averaged over 10 different active users and 10 different runs per active user.

Figures 21 to 23 show the results of evaluations for the flocks-of-agents based recommender system (FlockRecom - FR) in comparison to the results of collaborative filtering (CF) for 50 users, where $sim_{th} = 0.07$, $d_{th} = 0.4$, and $d_{ideal\_th} = 0.1$. The parameters were set by trial and error. The comparisons are based on the average precision, recall, and F1 values for varying n-values for top-n

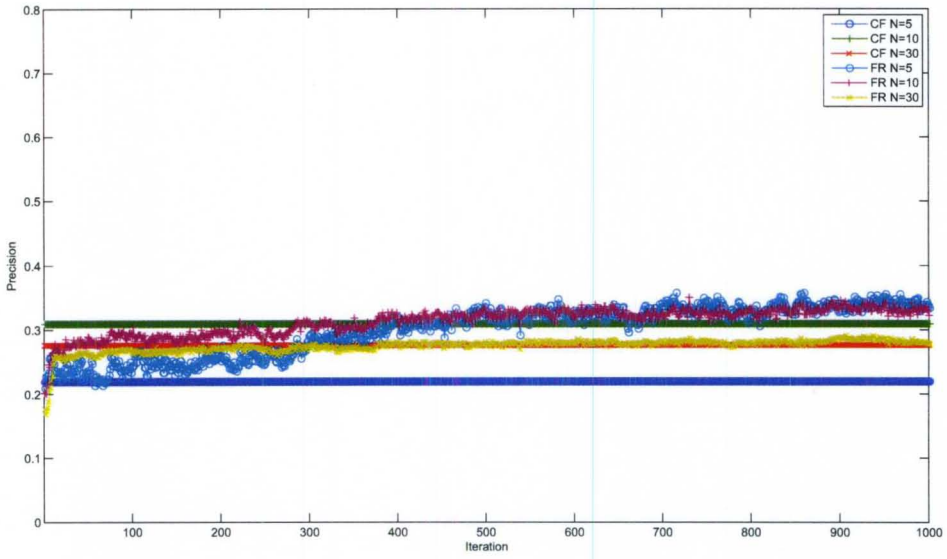


Figure 21. Flock-based recommender system (FR) compared to standard collaborative filtering (CF) based on precision.

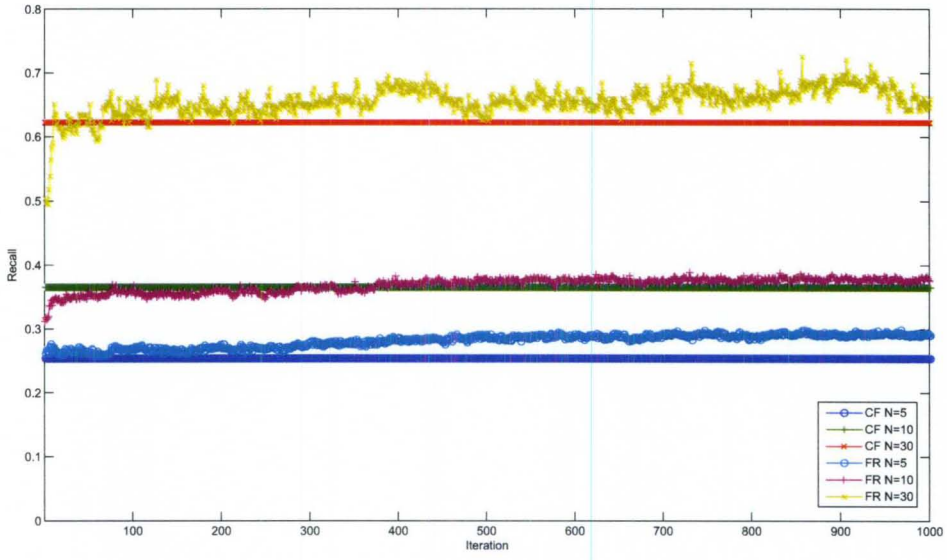


Figure 22. Flock-based recommender system (FR) compared to standard collaborative filtering (CF) based on recall.

recommendations, over time. The figures display the quality versus iteration or time averaged over all 10 users and 10 runs per active user.

From Figure 21, we observe that the precision values for FlockRecom are slightly better than those for CF, especially, for small N. Additionally, FlockRecom provides more variety in the

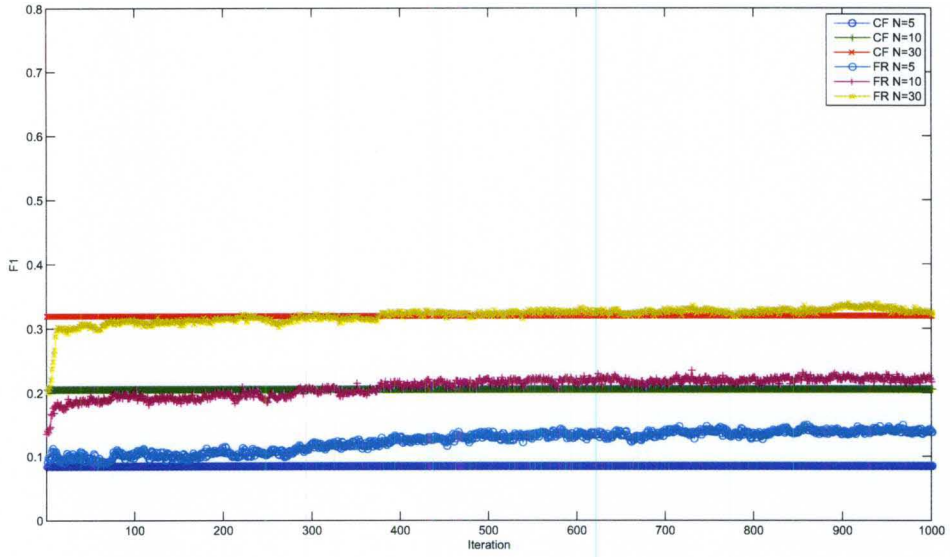


Figure 23. Flock-based recommender system (FR) compared to standard collaborative filtering (CF) based on F1.

recommendations. Similarly, as seen in Figure 22, FlockRecom produced slightly higher recall values than CF. As expected, both for FlockRecom and CF, recall values increased as N, the number of recommended items, was increased. As a result, the F1 metric was higher for FlockRecom, especially for $N = 5$, as Figure 23 shows.

The fluctuations in Figures 21 to 23 are due to the exploration in FlockRecom, thus showing that, unlike CF, FlockRecom does not recommended the same items over and over. Figure 24 presents the number of times each joke is recommended for a specific user over 10 different runs. While CF kept recommending the same items, FlockRecom added exploration and variety without losing from the precision, recall, and F1 quality.

To sum up, FlockRecom produced slightly better results than CF, after a sufficient number of iterations. For small values of N (which is preferred to avoid overloading users with too many recommendations), FlockRecom computed better recommendations, suggesting a more effective and realistic recommendation strategy. Morever, FlockRecom is more successful in exploration and in overcoming over-specialization.

Figures 25 to 27 show the results for a typical active user. The figures display the quality versus time in comparison with the traditional collaborative filtering (CF). In Figure 25, plots labeled FR (FlockRecom) show that precision gets significantly improved in the first 400 iterations and later keeps increasing slowly. As expected, when a smaller number of items are recommended, precision

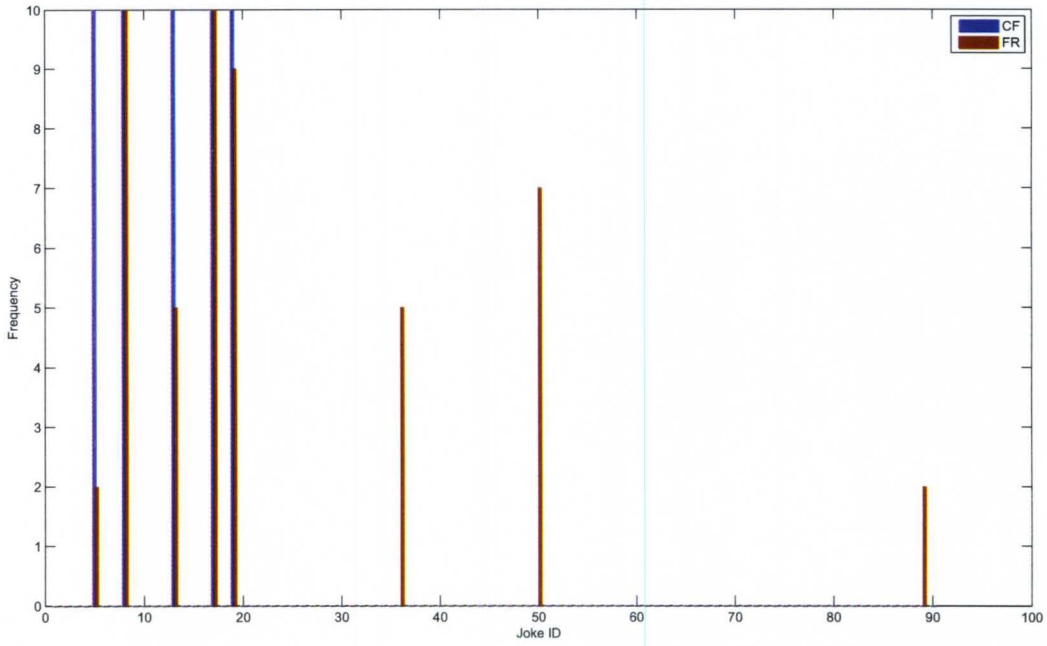


Figure 24. The variety in the recommended items of Flock-based recommender system (FR) compared to standard collaborative filtering-based recommender system (CF) for different numbers of top-5 recommendations, at iteration 100, 50 users. Averaged over 10 different runs per 1 active user. The x-axis represents the joke id.

was higher. Comparing FR to CF in Figure 25, we observe that precision values are better for FR for small N, and similar for FlockRecom and CF for bigger N. Moreover, FlockRecom provides more variety in the recommendations. Similarly, in FlockRecom, recall was improved with the number of iterations, as seen in Figure 26. For small N, FlockRecom produced better results. For big N, FlockRecom needed more iterations to compute similar recall values to CF. For both FlockRecom and CF, recall values increased as N, the number of recommended items, was increased, as shown in Figure 26.

To sum up, Figure 25, Figure 26, and Figure 27 show that, as the number of iterations increases, the neighborhood quality increases, thus the quality of the recommendations increases. We notice that FlockRecom continues to improve its recommendations, eventually reaching higher quality levels compared to standard CF. After enough number of iterations, FlockRecom produced better results than CF. Moreover, FlockRecom was more successful at exploration and overcoming over-specialization. These improvements are hence due solely to the dynamic nature of the flocking behavior of the agents that form dynamic neighborhoods that do not cause stagnation in the

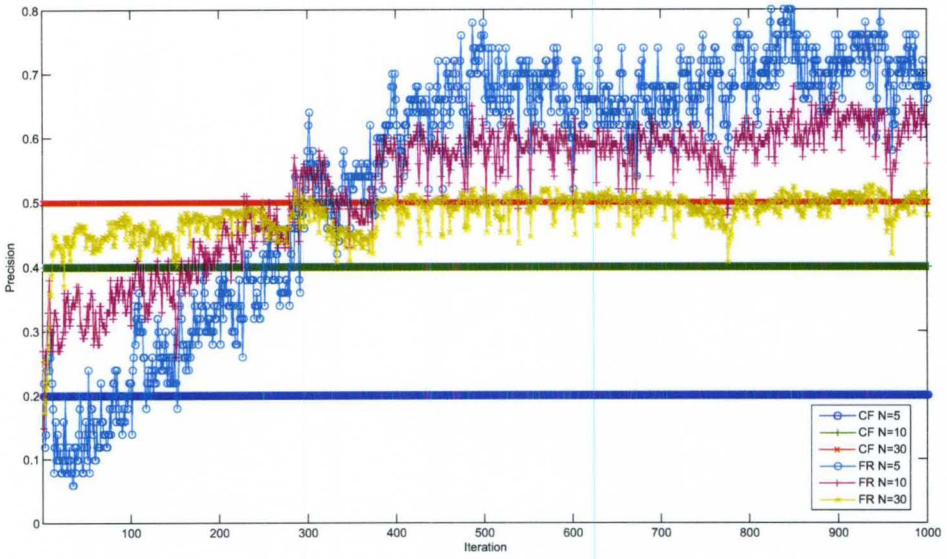


Figure 25. Flock-based recommender system (FR) compared to standard collaborative filtering-based recommender system (CF) based on the average precision values for different numbers of top-n recommendations, over time. Averaged over 1 active user, 10 different runs. The x-axis represents the iteration number.

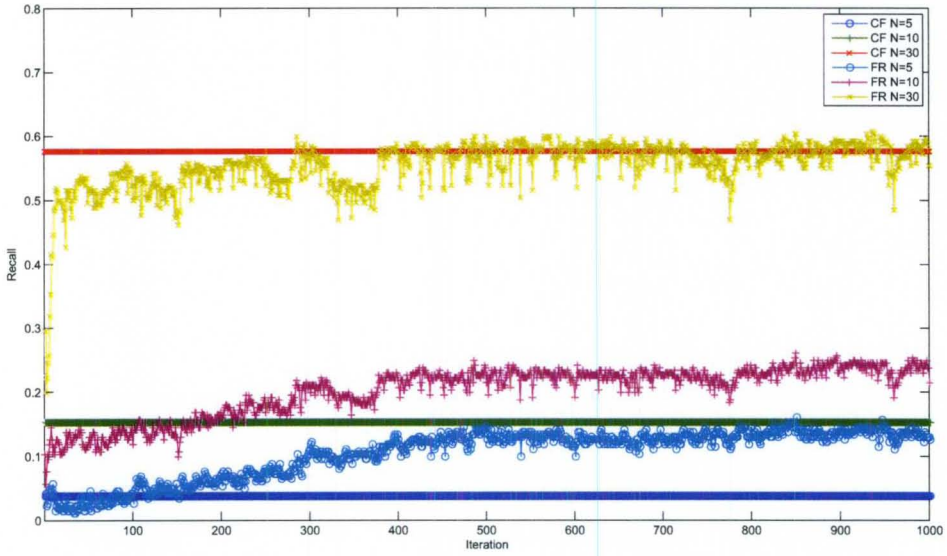


Figure 26. Flock-based recommender system (FR) compared to standard collaborative filtering-based recommender system (CF) based on the average recall values for different numbers of top-n recommendations, over time. Averaged over 1 active user, 10 different runs. The x-axis represents the iteration number.

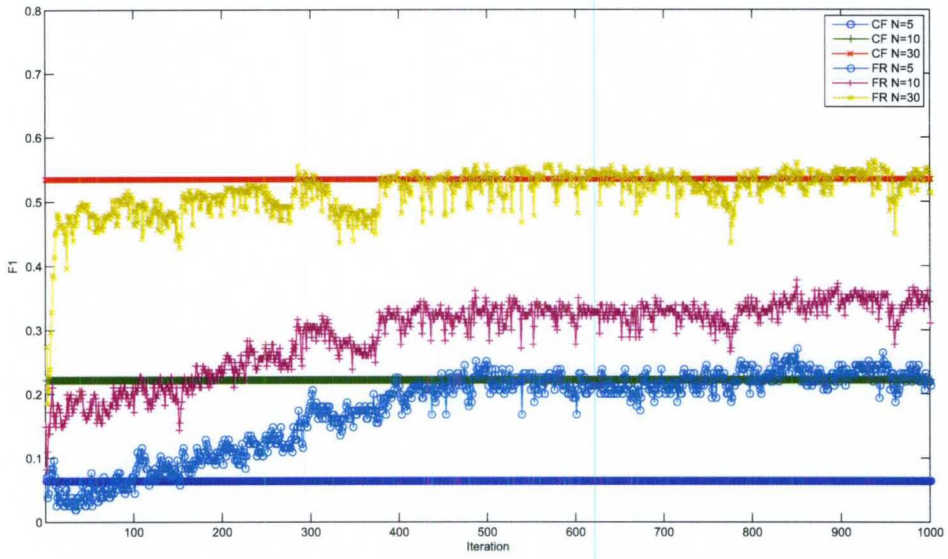


Figure 27. Flock-based recommender system (FR) compared to standard collaborative filtering-based recommender system (CF) based on the average F1 values for different numbers of top-n recommendations, over time. Averaged over 1 active user, 10 different runs. The x-axis represents the iteration number.

recommendations.

Table 20 shows the quality levels over several iterations for the two methods for 3 values of N. Note how FlockRecom clearly outperforms CF by "continuing" to learn and thus improving recommendations with time.

The next experiment expands the previous experiemnt that used the first 50 users, by using a bigger data set with 10 times as many users, thus totalling 500 users. Figures 28 to 30 show the results of evaluations for the flocks-of-agents based recommender system (FlockRecom - FR) in comparison to the results of collaborative filtering (CF) based on the first 500 users, where $sim_{th} = 0.07$, $d_{th} = 0.4$, and $d_{ideal\_th} = 0.1$. The parameters were set by trial and error. The comparisons are based on the average precision, recall, and F1 values for varying n-values for top-n recommendations, over time. The figures display the quality versus iteration or time averaged over all 10 users and 10 runs per active user.

From Figure 28, we observe that the precision values for FlockRecom become slightly better than those for CF, starting after iteration 200 and reach their highest values around 500 iterations. Additionally, FlockRecom provides more variety in the recommendations, as will be shown more clearly later. Similarly, as seen in Figure 29, FlockRecom produced slightly higher recall values

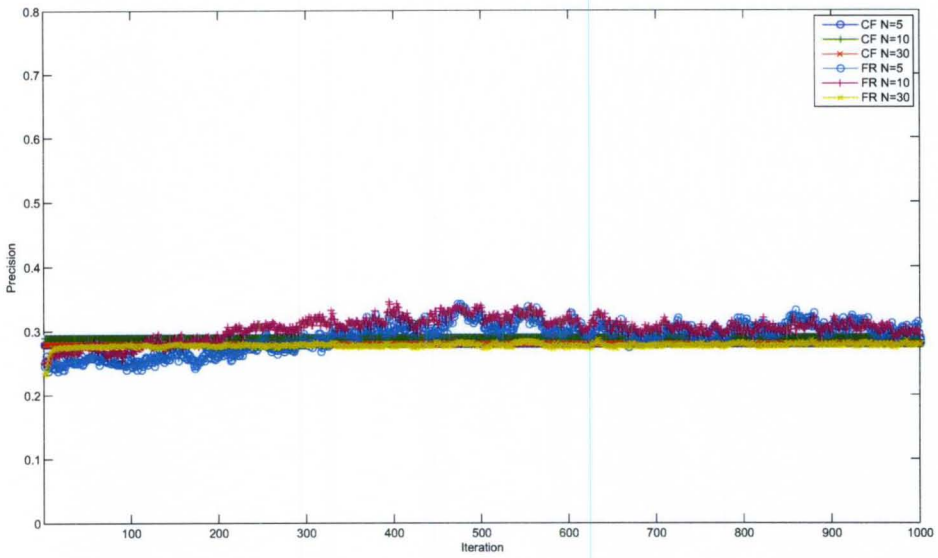


Figure 28. Flock-based recommender system (FR) compared to standard collaborative filtering (CF) based on precision, 500 users.

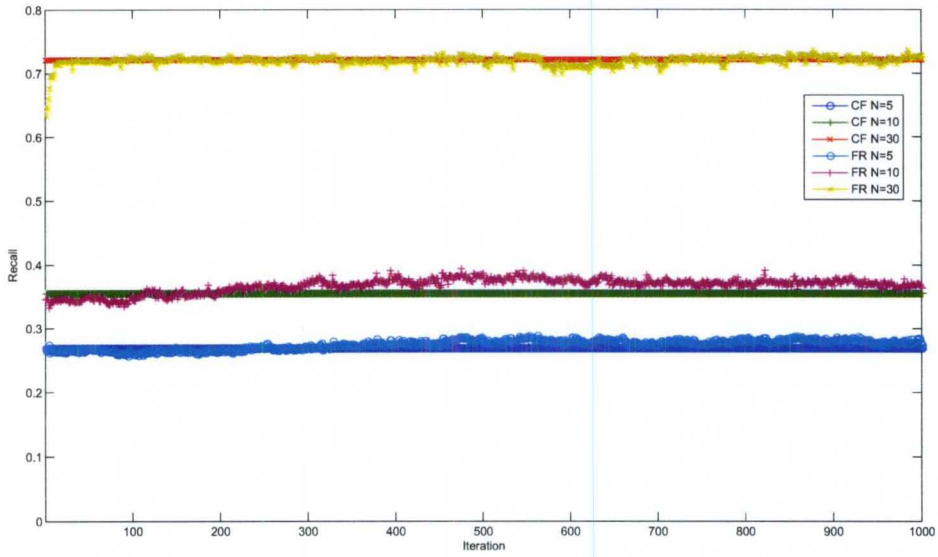


Figure 29. Flock-based recommender system (FR) compared to standard collaborative filtering (CF) based on recall, 500 users.

than CF, especially for small N (5 and 10), which are the realistic values for reasonable real life recommender systems. As expected, both for FlockRecom and CF, recall values increased as N, the number of recommended items, was increased. As a result, the F1 metric was higher for FlockRecom, especially for $N = 5$, as Figure 30 shows.

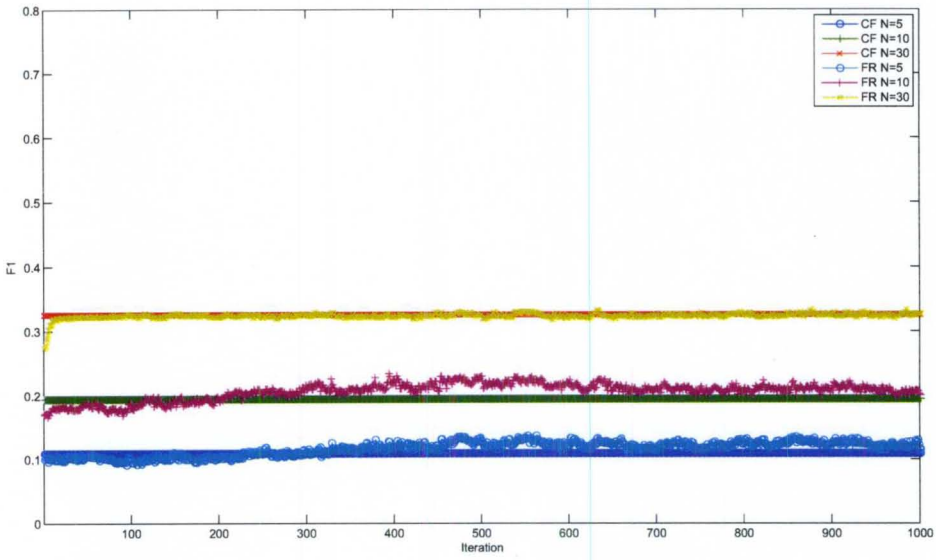


Figure 30. Flock-based recommender system (FR) compared to standard collaborative filtering (CF) based on F1, 500 users.

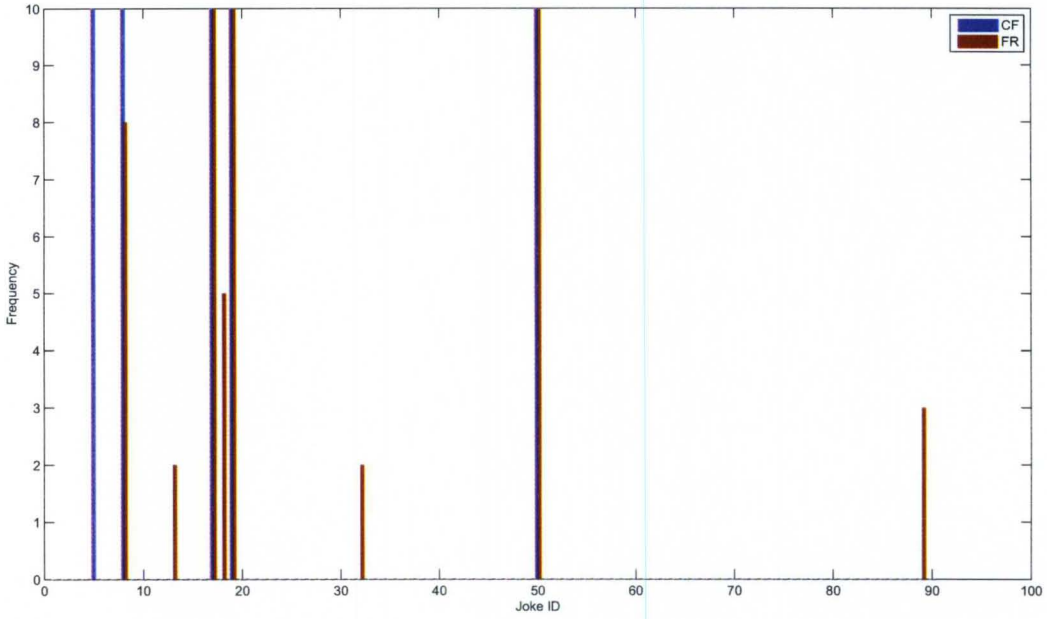


Figure 31. The variety in the recommended items of Flock-based recommender system (FR) compared to standard collaborative filtering-based recommender system (CF) for different numbers of top-5 recommendations, at iteration 100, 500 users. Averaged over 10 different runs per 1 active user. The x-axis represents the joke id.

The fluctuations in Figures 28 to 30 are due to the exploration in FlockRecom, thus showing that, unlike CF, FlockRecom does not recommended the same items over and over. To further verify this desirable characteristic, Figure 31 presents the number of times each joke is recommended for a specific user over 10 different runs. While CF kept recommending the same items, FlockRecom added exploration and variety without losing from the precision, recall, and F1 quality.

4.4 Conclusion

In this research, a new recommender system approach called the flocks-of-agents based recommender system (FlockRecom) was presented. This new approach is based on swarm intelligence, specifically, the dynamic collaboration between bird flocks in nature. The results were compared to the traditional user-based nearest neighbor collaborative filtering. Although the implementation and parameter setting is easier in collaborative filtering, FlockRecom was more successful at providing variety in the recommendations without losing recommendation quality. One problem suffered by some recommender systems is over-specialization. When the recommendations are limited to the user's behavior or user's profile, the user can be restricted to seeing only similar items, and there will be no randomness. In artificial intelligence, this problem is known as the exploration/exploitation dilemma. Although collaborative filtering can counteract over-specialization by suggesting different items, the dynamic structure of the FlockRecom algorithm makes it more successful at solving the exploration/exploitation dilemma, which is also practically observed in the experimental results.

TABLE 20. The quality levels averaged over 10 runs of 1 active user at several iterations for the FlockRecom and CF at 3 values of N.

FlockRecom						**CF**						
N=5		**N=10**		**N=30**		**N=5**		**N=10**		**N=30**		**Iter.**
Prec.	**Rec.**	**Prec.**	**Rec.**	**Prec.**	**Rec.**	**Prec.**	**Rec.**	**Prec.**	**Rec.**	**Prec.**	**Rec.**	
0.28	0.05	0.42	0.16	0.46	0.54	0.20	0.04	0.4	0.15	0.5	0.58	200
0.62	0.12	0.57	0.22	0.50	0.58	0.20	0.04	0.4	0.15	0.5	0.58	400
0.84	0.16	0.68	0.26	0.52	0.60	0.20	0.04	0.4	0.15	0.5	0.58	850

CHAPTER 5

A SWARM BASED APPROACH FOR DYNAMIC DATA CLUSTERING

The dynamism of life reflects on information generation and empowers the information overload problem. Our previous work using a special type of swarms, known as flocks of agents, provided algorithms for clustering high-dimensional sparse data and evaluations were performed on several UCI machine learning data sets and Web usage session data [128, 129]. However, dynamic domains, such as practically any data generated on the Web, may require frequent costly updates of the clusters (and the visualization), whenever new data records are added to the dataset. Additionally, change in the data records may result in a change of clustering over time. An example of this is the change in the interest of users of a given Web service. Therefore, clusters may need to be updated, thus leading to the need to mine dynamic clusters. Moreover, when the data is high-dimensional and sparse, the dynamic behavior makes an already difficult problem even more challenging. This chapter classifies different clustering paradigms related to dynamic clustering, then discusses the distinctions between dynamic data clustering and other paradigms, and describes a Dynamic-FClust Algorithm, which is based on flocks of agents as a biological metaphor. This algorithm falls within the swarm-based clustering family, which is unique compared to other approaches, because its model is an ongoing swarm of agents that socially interact with each other, and is therefore inherently dynamic. Experiments on real and artificial data illustrate the workings of the dynamic algorithm and its advantages over its FClust baseline.

5.1 Introduction

As defined earlier, *clustering* is the problem of finding groups in a dataset, according to some data properties and attributes which have a meaning in some context [60, 59]. Depending on the characteristics of the dataset and the goal of the knowledge discovery process, different clustering paradigms have emerged, namely static (or conventional), incremental, and stream [43, 30, 9]. And

more recently, we saw an interest in dynamic data clustering which concentrates not only on concept drift but also on data items that change in time.

Unlike conventional clustering, in *dynamic clustering*, data is collected continuously over time, thus the whole data set is not available initially. *Dynamic* domains, such as practically any data generated on the Web, may require frequent costly updates of the clusters (and the visualization), whenever new data records are added to the dataset. The new coming data may be due to new user activity on a website (clickstreams) or a search engine (queries), or new Web pages in the case of document clustering, etc. Additionally, a concept drift in the data records may result in a change of clustering in time [137]. An example of this is the change in the interest of users on an online service. Therefore, clusters may need to be updated, thus leading to the need to mine dynamic clusters. When the data is moreover high-dimensional and sparse, the dynamic behavior makes an already difficult problem even more challenging. Swarm intelligence methods, mainly ant-based systems, have been used as well as several other methods including repeatedly running a given clustering algorithm, such as K-means, at each time step [122, 79]. Incremental clustering methods can also be adapted to this task, but they mostly ignore the dynamic nature of the data, which requires distinguishing between old and new data. Similarly, stream clustering methods can be used for this task, but they mostly emphasize not storing the whole dataset and thus cannot handle dynamic data *records*. This chapter describes our design of a simultaneous clustering and visualization algorithm for dynamic data and proposes the Dynamic-FClust Algorithm, which is based on flocks of agents as a biological metaphor. As a swarm-based algorithm our algorithm distinguishes itself from most other approaches by learning a model in the form of an ongoing swarm of agents that socially interact with each other, and is therefore inherently dynamic. A flock-based model has clear advantages over the particle swarm model, because the agents in a set of flocks are inherently associated with data, and are thus directly amenable to visualization tasks; whereas particle swarms tend to not be associated with data, but rather with candidate solutions to an optimization criterion. Moreover, the dynamic structure of the algorithm and the social interaction between agents provides a suitable foundation for modeling dynamic human behavior on the Web with many interesting applications such as creating dynamic online recommender systems.

This chapter starts by explaining the dynamic data clustering paradigm in Section 5.2 and comparing it with other clustering paradigms in Section 5.2.1. Then, we present a new dynamic clustering algorithm *Dynamic-FClust* in Section 5.3. In Section 5.4, we describe our contributions so far, then present our experiments and evaluation techniques. Finally, we make our conclusions

and discuss future work in Section 5.5.

5.2 Dynamic Data Clustering Paradigm

Given the clustering definition in Section 5.1, we define *dynamic data clustering* (in short, dynamic clustering) as follows: *Extracting clusters in a dataset in which new data records may be added to the dataset, some data records may diminish with time and may even be removed from the dataset, while other data records may change in time.* Moreover, in the case of a concept drift [137], a dynamic clustering algorithm should have the capability to handle generating new clusters and follow evolving clusters as needed. (See Section 5.3 for more details on our approach.)

The characteristics of the dynamic data clustering process are:

- **The ability to cluster the available data:** This is in fact the basic definition of static (conventional) clustering . If a batch of data is available to the application, we expect the system to form groups of similar items after an acceptable number of iterations.
- **Cluster Number Extraction:** It is expected that the number of clusters is not given as an input to the algorithm since it can change with time especially.
- **When new data items are added, avoid clustering the updated dataset from scratch:** When new data batches are available, the algorithm should use the clustering information from the previous state and produce clustering results for the updated dataset. In other words, it is not expected to re-start from scratch whenever new data arrives.
- **The ability to handle emerging and disappearing clusters:** With the addition of new data items, new clusters may emerge and in that case, the application should create new clusters in an acceptable number of iterations. Additionally, some old clusters may vanish and the algorithm is expected to update the data clusters in such a case.
- **Dynamic Data Records:** A *dynamic data record* is a data record whose attributes may change with time. An example is a user on a web site, whose likes and dislikes may change in time. A dynamic clustering algorithm should consider the existence of dynamic data records and update clustering results as the data record changes.

5.2.1 Differences Between Dynamic Clustering and Other Clustering Paradigms

Given the dynamic clustering paradigm in Section 5.2, a classification of clustering paradigms is presented in this section.

The most common and most studied paradigm is conventional a.k.a static clustering [60, 85, 30, 129], where the goal is to cluster a dataset that is fully available before and during the clustering process.

Another paradigm is known as *incremental clustering* or *online clustering* in which, data becomes available in batches over time [43, 7, 32, 79]. Incremental clustering was initially proposed as a solution to clustering on very large databases [43] and unlike dynamic clustering, does not consider change in "existing" data items. Moreover, in some application, the whole dataset may be available initially. However, if the dataset is huge, batches are formed by sampling, and clustering is performed incrementally [43]. IncrementalDBSCAN can handle emerging and disappearing clusters, but does not handle data items that change in time [43]. One common approach for incremental clustering is to repeatedly apply a classical clustering algorithm every time more data arrives or change in data occurs, in which case a high computational cost may be incurred. Additionally, such an approach does not differentiate new data from existing data. In another algorithm, called leader-follower clustering (LFC), a K-means type clustering is used [42]. In another approach, clusters are generated according to the initially available data batch and as the new data arrives, new data is assigned to existing clusters [32]. Ant colony-based swarm intelligence algorithms are some of the most common approaches for solving incremental clustering in SI [122, 79, 32].

The next paradigm is the stream clustering paradigm, where a stream of data is modeled preferably in one-pass, i.e. by reading each data item only once [9, 96], and the aging or disappearing clusters may be handled via a forgetting mechanism. Data may be stored partially for a window frame. Stream clustering methods can be used for clustering datasets with concept drift, but they generally do not store the whole dataset and thus cannot handle dynamic data items.

Table 21 gives a comparison of different paradigms and Table 22 gives sample approaches from these different paradigms. To sum up, many clustering algorithms are unable to handle dynamic data items. Moreover, most approaches are unable to perform clustering and visualization on a continuous basis or to have new data-items self-organize into the existing clusters (or into new clusters if necessary), unless a new and additional processing and analysis phase is applied. This disadvantage is also present in more recent approaches using Self-Organizing Maps, as in Kohonen maps [122]. Therefore, while a benchmark comparison of the above cited methods with Dynamic-

FClust should be interesting to explore, any serious comparison will be complicated by the ability of the dynamic FClust algorithm to a) handle dynamic data items, b) perform continuous mappings with simultaneous continuous visualization compared to the inability of the remaining techniques to

TABLE 21. Comparison of different clustering paradigms.

Paradigm Paradigm	**Requires All Data**	**Stores Data**	**Emerging Clusters**	**Disappearing Clusters**	**Evolution Tracking**	**Dynamic Record**	**Sample Dataset**
Static	Yes	Yes	No	No	No	No	Iris Dataset[5]
Incremental	No	Yes	Maybe	No	No	No	Birch Dataset[150]
Stream	No	No	Yes	Possible (via forgetting)	Possible	No	KDD Cup 1999 Dataset[3]
Dynamic	No	Yes	Yes	Yes	Yes	Yes	Movielens Dataset[4]

TABLE 22. Comparison of different clustering and visualization algorithms.

Algorithm	Incremental Clustering	Emerging/ Disappearing Clusters	Changing Data Items	Cluster No Extraction	Visualization	Dynamic Visualization	Typically Compared to	Paradigm
KMeans [85]	No	No	No	No	No	No	All	Static
IncrementalDBSCAN [43]	Yes	Yes	No	Yes	No	No	DBSCAN	Incremental
Incremental Ant Clustering [122]	Yes	Yes	No	No	No	No	Static Ant Clustering	Incremental
LFC [42]	Yes	Only Emerging	No	Yes	No	No	K-Means	Incremental
SOM [88, 87, 67]	No	No	No	Yes	Yes	No	NA	Static
Incremental SOM [21]	Yes	Only Emerging	No	Yes	No	No	SOM	Incremental
CluStream [9]	Yes	Yes	No	Yes	No	No	STREAM	Stream
FClust[117]	No	No	No	Yes	Yes	No	K-Means	Static
Dynamic-FClust	Yes	Yes	Yes	Yes	Yes	Yes	FClust	Dynamic

accomplish them both.

5.3 Swarm-based Dynamic Clustering Algorithm

The clustering goals defined in Section 5.2 are specialized for dynamic clustering [60, 122]. Additionally, any flock-based clustering approach has the simultaneous clustering and visualization goal, which makes dynamic-FClust naturally suitable for cluster evolution tracking.

Although "dynamic items" were studied in different problem domains such as classification, information retrieval, and recommender systems, (ex: Rocchio Classification [86]), to the best of our knowledge, there have been no studies on dynamic clustering with "dynamic items", i.e. data items changing in time. Thus, although there are applications capable of handling some of the above expectations of dynamic data clustering given in Section 5.2, to the best of our knowledge, there are no existing algorithms that can satisfy all the above expectations.

In the original *FClust* algorithm, it was assumed that the dataset was fully available and static, whereas in the proposed *Dynamic-FClust* algorithm, data entries are observed one at a time. Moreover, information regarding a data item may be updated in different time steps. After a small number of iterations, the initial clusters are formed, then the clusters continue to be updated with each new coming record or update to a record.

Algorithm 17 gives a detailed explanation of Dynamic-FClust. At each iteration, if data removal is expected in the problem domain, we start by removing the agents that represent data records that have been removed. Then, we assume that one new data entry is read per iteration. If the data is related to a data item, which is not represented by an agent on the system, a new agent is created and initialized in Line 15 to Line 17. In the next step, the ideal distances of the modified agents are updated. Note that in the Algorithm, the velocity vector $\overline{v}$, is a unit vector, (i.e. $||\overline{v}|| = 1$), representing the direction of an agent. Later, for each agent i, the neighboring agents that are close enough to i on the visualization panel, are extracted in Line 25, where *d(i,j)* is the 2D Euclidean distance between agents i and j. Then, in line 27, the velocity effect on i due to neighbor j is computed where $\overline{v}_{cap}(i,j)$ is the unit vector pointing from i to j and the coefficient $\beta(i,j)$ is computed using Algorithm 18. For each neighbor, three cases arise as shown in Algorithm 18.

Case 1: If the distance between the agents i and j is equal to the ideal distance between them (Line 1), there is no attempt to change i's velocity due to j (Line 2).

Case 2: If the distance between the agents i and j is greater than the ideal distance between them (Line 3), an attraction force will move i closer to j, with a more similar velocity to j (Line 4).

Algorithm 17 The Flocks of Agent-based Dynamic Clustering Algorithm **Dynamic-FClust**

Input: Dataset; $data_d$:Expired data; $data_a$:added data

Output: Dynamic visualization of interaction between the data items. Agents corresponding to more similar items are located closer in the 2D visualization panel. Extracted clusters (on request).

repeat
  repeat
    if A data record $data_d$ has expired **then**
      Remove the agent d corresponding to $data_d$ from the visualization panel.
      Delete $data_d$ from the dataset.
    end if
  until There is no expired data record
  repeat
    Read data entry related to data record $data_a$
    if $data_a$ does not exist in the existing data **then**
      if Maximum number of agents is reached **then**
        Choose an agent to discard.
        Remove the data record corresponding to this agent.
      end if
      Create a new agent and map $data_a$ to agent a.
      Place agent a on the visualization panel (e.g. randomly or based on domain knowledge).
      Initialize the velocity of agent a (e.g. randomly or based on domain knowledge).
    else // Data record exists and will be updated
      Update the data record with the new data entry.
      Find agent a that represents $data_a$.
    end if
    Update the ideal distances, d_{ideal}, between a and the rest of the agents based on their associated data in the original data space.
  until There is no new data entry
  for each agent i **do**
    for all j such that $d(j,i) \leq d_{th}$ and $i \neq j$ **do**
      Call Algorithm 18 (compute β).
      $\overline{v}_{resulting}(i,j) \leftarrow \overline{v}(j) + \beta(i,j) \times \overline{v}_{cap}(i,j)$
    end for
    if $\exists\ j$ such that $d(j,i) \leq d_{th}$ and $i \neq j$ **then**
      $\overline{w}(i) = \text{normalize}\ (\sum_{j|d(j,i)\leq d_{th}\&i\neq j} \overline{v}_{resulting}(i,j))$
      if The angle between $\overline{v}(i)$ and $\overline{w}(i)$ is less than or equal to 90 degrees **then**
        $\overline{v}_{next}(i) \leftarrow \overline{w}(i)$
      else
        $\overline{v}_{next}(i) \leftarrow \overline{v}(i)$
      end if
    else
      $\overline{v}_{next}(i) \leftarrow \overline{v}(i)$
    end if
    $amp_{next}(\text{i}) \leftarrow amp_{def} + \frac{d_{ideal\_th}}{20\times(neighbor\_no(i)+1)}$
  end for
  for each agent i **do**
    compute new position $p_{next}(i) \leftarrow p_{current}(i) + amp_{next}(i) \times \overline{v}_{next}(i)$
  end for
  Move all agents to the updated positions and update current velocities.
  Update visualization screen with the updated positions and updated velocities.
  if Clustering results are requested **then**
    Extract Clusters using Algorithm 9.
  end if
until System is stopped.

Algorithm 18 Compute Beta

Input: Agents i and j.
Output: $\beta(i,j) \in R$.

1: **if** $d(i,j) = d_{ideal}(i,j)$ **then**
2: $\quad \beta(i,j) \leftarrow 0$
3: **else if** $d(i,j) > d_{ideal}(i,j)$ **then** // attraction
4: $\quad \beta(i,j) \leftarrow 4 \times \left(\frac{d(i,j)-d_{ideal}(i,j)}{d_{th}-d_{ideal}(i,j)}\right)^2$
5: **else** // repulsion
6: $\quad \beta(i,j) \leftarrow -4 \times \left(1 - \frac{d(i,j))}{d_{ideal}(i,j)}\right)^2$
7: **end if**
8: **return** $\beta(i,j)$

Case 3: If the distance between the agents i and j is smaller than the ideal distance between them (Algorithm 18, Line 5), a repelling force will move i away from j, with a less similar velocity to j (Line 6). Next is the computation of the updated velocity of agent i, $\overline{v}_{next}(i)$, between lines 29 and 38. First, if i has neighbors, then their resulting velocities on i are added and normalized. If the total, normalized velocity $\overline{w}$, does not change the agent's current direction more than 90 degrees, then the updated velocity is set to $\overline{w}$. Otherwise, the velocity is kept unchanged for the next iteration. Similarly, if agent i does not have any neighbors -note that an agent is not a neighbor of itself- then the velocity will be kept the same.

In the algorithm, amp_{def} is the default minimum amplitude which is empirically set to $\frac{1}{5} \times d_{ideal\_th}$ and the computation of the coefficient $\beta(i,j)$ in Algorithm 18 is from the existing literature [117].

Below, we present our analysis of the design constraints and provide guidelines that need to be considered.

Design guidelines and constraints:

- **Number of agents on visualization panel:** Computational and main memory constraints impose a limitation on the number of agents. In addition, too many agents may cause cluttering in the visualization panel.
- **Amount of data that is represented in the visualization panel**: Not only do computational and main memory constraints impose a limitation on the number of agents, but a limit on the age of the data that should be visualized may be required (e.g. visualizing only data that arrived during the last 24 hours).

In order to observe the above guidelines, we observe that we may replace some data items when the upper bound on the number of agents is reached. This opens three possibilities that need to

be weighed very carefully in light of the specific context and goals of the clustering and visualization. The three options, which may be applied exclusively of one another, using a priority rule, or even mixed together in a stochastic strategy, are:

1. Always replace the oldest data item.

2. Always replace the least interesting (e.g. redundant) data item. The notion of *interesting* data is crucial here, since it is not always associated with unique or non-redundant data. For example in the case of a query dataset, the *interesting* data items may be the sessions with *more profitable* queries, the sessions including the keywords with the highest bid values, the sessions related to e-commerce search or to emergency situations, etc.

3. Always replace one of the outlying data items.

5.3.1 The Main Computational Constraint

In dynamic clustering algorithms, it is expected that the clustering of existing data items is done by the time new data arrives. Let $T_{c_{t_i}}$ be the time needed to cluster existing data items at time t_i, the next data batch arrive at time t_{i+1} and $\Delta t_i = t_{i+1} - t_i$.

$$T_{c_{t_i}} \leq \Delta t_i \quad \Rightarrow \quad \text{There is no problem with the above timing constraint.}$$
$$T_{c_{t_i}} > \Delta t_i \quad \Rightarrow \quad \text{This violates the above timing constraint.}$$

In the case where the clustering of existing items can be finalized by the time a new data batch arrives, there is no problem. Otherwise, some alternative options should be considered to handle this problem:

- **Sampling:** Instead of comparing all neighboring agents, a subset of neighbors can be compared to update the velocity and amplitude. How many agents are going to be sampled depends on the problem and requires further study. The selection of agents can be done randomly or k-nearest neighbors can be selected.

- **Improved initialization:** Initial location of the agents affects the convergence time [117]. Some heuristics for initialization of location and velocity can be developed. When a 2 dimensional visualization panel is considered:

  - If the data is 2 dimensional and has numerical attributes:

TABLE 23

Comparison of alternative solutions to satisfy computational constraints.

Solution	Advantage	Disadvantage
Sampling	Decreases the computational cost.	May decrease the accuracy.
Improved initialization	May increase the accuracy.	Increases the computational cost.
Delayed input	Easy implementation.	Does not guarantee a solution.

    * Data can be normalized to be of unit length and agent location can be assigned to be the normalized data record.
    * The data can be linearly normalized to $[0, 1]$ and agent locations can be assigned to be the normalized data record.
    * Data can be normalized to be of unit length and agent velocities can be assigned to be the normalized data record.
    * The data can be linearly normalized to $[0, 1]$ and agent locations can be assigned to be a unit vector computed from the normalized data record.
  - If the data has more than 2 dimensions, 2 dimensions can be selected by
    * Expert knowledge
    * Data analysis techniques such as principal component analysis and multiple correspondence analysis.
  - The agent velocities can be oriented in a centrifugal way with respect to the center of the 2D environment [117].

- **Delayed input:** Instead of adding every new batch as it arrives, the addition of a new batch may be delayed for δt iterations, where $0 \leq \delta t \leq$ the time between the current new data batch and next new data batch.

Table 23 gives a comparison of different options if the clustering of existing items is not finalized by the time a new data batch arrives. Different solution or hybrid-solutions may be applied depending on the dataset.

5.3.2 Main Memory Constraints

Main memory constraints impose a limitation on the number of agents. Let N_a be the number of agents and $N_{a_{max}}$ be the maximum number of agents that can be loaded, predetermined

based on the available memory space to be allocated. This opens three possibilities that need to be weighed very carefully in light of the specific context and goals of the clustering and visualization. The three options, which may be applied either exclusively of one another, or using a priority rule, or even mixed together in a stochastic strategy, are:

1. Always replace the agent that is mapped to the oldest data item.

2. Always replace the least interesting (e.g. redundant) data item. The notion of *interesting* data is crucial here, since it is not always associated with unique or non-redundant data. For a web search query log dataset, the *interesting* data items may be the sessions with *more profitable* queries, the sessions including the keywords with the highest bid values, the sessions related to e-commerce search, etc.

3. Always replace one of the outlying data items. An outlying data item can be determined as the data item which is mapped to the agent with the minimum number of neighboring agents.

If removing the outlying items is adopted, this would increase the computational time required to determine these outlying agents. The definition of "interesting" deeply affects the efficiency of removing the least interesting item.

5.3.3 Setting the Parameters for Dynamic-FClust

The selection of parameters has a big impact on the convergence of the Dynamic-FClust algorithm. The first parameter is d_{th}, the distance threshold, which defines the neighborhood size (see line 25 in Algorithm 17). When d_{th} is too small, the agents cannot affect each other, and when it is too high, the algorithm may not converge. Another parameter d_{ideal_th} affects the ideal distance via Equation (28) and the amplitude of the velocity (see line 39 in Algorithm 17). When d_{ideal_th} is too small, the agents position themselves too close to each other on the visualization panel, visualization of clusters may be harder, and the clustering process may require too many iterations since the amplitude of the motion is not high enough. On the other hand, when d_{ideal_th} is too high, even similar agents may have big ideal distances, agents do not attract each other, and thus the algorithm does not converge. One method to compute the ideal distance between two agents, d_{ideal}, is given in Equation (28). If the ideal distances are overestimated, then the clusters cannot be observed on the visualization panel.

The similarity threshold, sim_{th}, in (28) may be updated incrementally via Equation (26), which is a flexible way to tune the similarity threshold, by a tuning factor α, where $1 \leq \alpha \leq$

$\frac{Max\{sim(i,j)\}}{\overline{sim}}$, when $\overline{sim}$ is the average of $sim(i,j)$. Or, the sim_{th} can first be given as an input to the clustering algorithm based on the average in the initial batch of data, and then updated periodically. If the similarity threshold is too large, then the algorithm will fail to converge, whereas if it is too small, then different clusters risk being combined into one cluster.

5.3.4 Cluster Formation in Dynamic-FClust

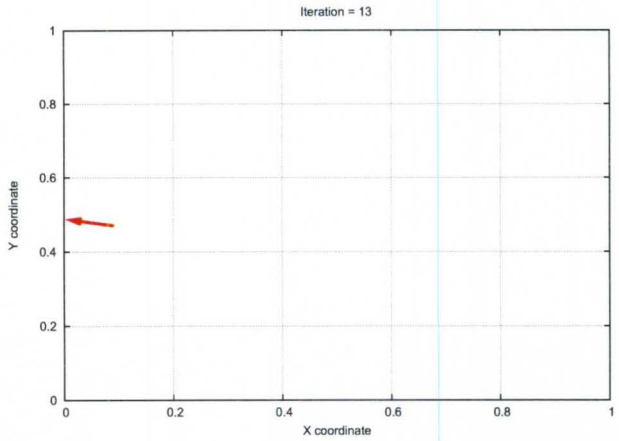
Cluster formation can be determined via different methods such as human experts, as well as entropy and ideal distance error calculations [129, 128, 117]. This is because when Dynamic-FClust is performed, flocks of agents are visually observable. However, the clusters are not explicitly formed and the data is not yet assigned to clusters. One method of forming clusters is using Human experts. The person marks the clusters and assigns agents to the clusters. Since there is a 1-to-1 mapping between agents and data records, the data will also end up being clustered. In addition to this, an automated procedure was presented in [117], which is given in Algorithm 9. Basically, a new cluster is created for an unlabeled agent. Then the neighboring agents of this cluster are explored, and all the agents which are similar to at least one of the agents in the cluster are inserted into this cluster. New agents are inserted to the cluster until no more agents can be inserted. Then the procedure restarts by creating another new cluster, and stops when all the agents are labeled. After cluster formation, a post processing phase is needed to cluster the original (input) data, to validate the results, and if possible to interpret the clusters.

5.3.5 Teleportation in FClust and Effect on Cluster Formation

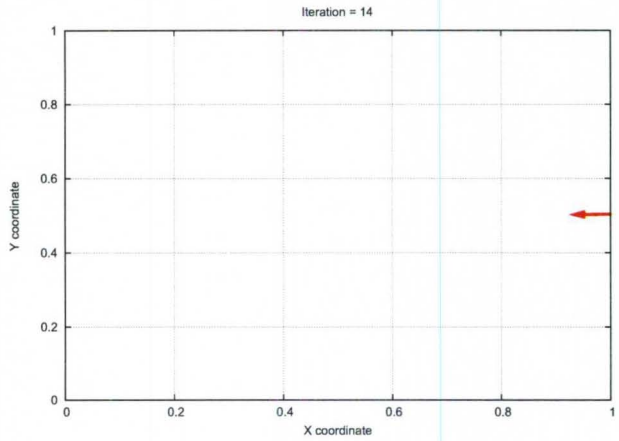
In previous research using FClust, the 2D Euclidean distance between agents on the visualization panel has been used to determine the distance in Algorithm 9, step 5. However, this computation may split clusters at the borders of the panel: When an agent is traveling on the visualization panel, it reaches the borders of the visualization panel from time to time. In that condition, the agent passes the border and continues to move from the other side which brings into our mind the concept of teleportation. The word teleportation is defined as a hypothetical mode of instantaneous transportation in the dictionary [1]. Thus we define teleportation in FClust as follows:

Definition 2 *Teleportation in FClust is the mode of instantaneous transportation that occurs when an agent crosses the border of the visualization panel and continues from the opposite side of the visualization panel.*

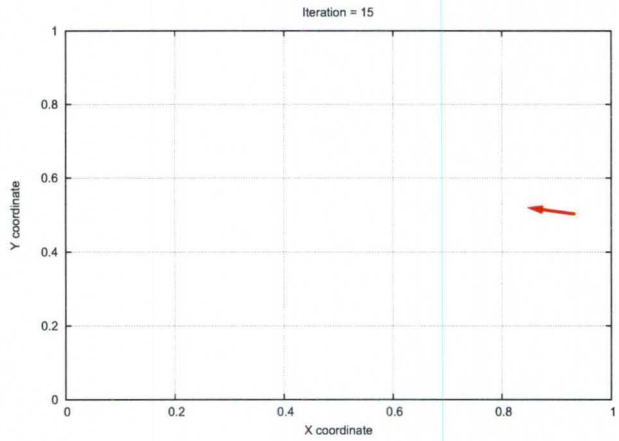
[1] Webster's online dictionary



(a) Before teleportation.



(b) During teleportation.



(c) After teleportation.

Figure 32. Teleportation illustrated by one agent (red arrow showing direction of movement) teleported from the left border to the right border.

Figure 32 illustrates an agent teleporting from the left border to the right border of the panel. Due to teleportation, a part of the cluster may pass the border and continue from the other

side, in which case, the 2D Euclidean distance would become too big to consider the two separated parts as one cluster and the cluster gets split. To solve this problem, we first define a modified 2D wrap-around distance in Equation (38) and then use a modified wrap-around cluster formation, which uses this 2D wrap-around distance, instead of the 2D Euclidean distance, thus compensating for the agent teleportation effect.

Given that each agent a_i's location, in the 2 dimensional real number space R^2, is represented by a vector x_i, y_i, where $0 \leq x_i \leq 1, 0 \leq y_i \leq 1$, the 2D wrap-around distance between two agents a_i and a_j is given by Equation (38).

$$d\_wrapping(a_i, a_j) = min \begin{cases} \sqrt{(x_i - x_j)^2 + (y_i - .y_j)^2}, & \text{no wrap-arround} \\ \sqrt{(1 - |x_i - x_j|)^2 + (y_i - .y_j)^2}, & \text{vertical borders connected} \\ \sqrt{(x_i - x_j)^2 + (1 - |y_i - .y_j|)^2}, & \text{horizontal borders connected} \\ \sqrt{(1 - |x_i - x_j|)^2 + (1 - |y_i - .y_j|)^2}, & \text{both borders connected} \end{cases} \tag{38}$$

The modified wrap-around cluster formation algorithm, which uses the wrap-around distance to define the neighborhood, is given in Algorithm 19.

Algorithm 19 Wrap-around Cluster Formation Algorithm

Input: Agents' coordinates.
Output: Clusters of agents.

```
for each agent i do
  if i's cluster is not assigned then
    Form a new cluster c
    Assign i to c
    for all agent j such that there exists an agent k ∈ c and d_wrap-around(k,j)≤ d_th and
    sim(k, j) > sim_th do
      Assign j to c
    end for
  end if
end for
```

5.3.6 Complexity Analysis of Dynamic-FClust

The Dynamic-FClust algorithm, given in Algorithm 17, compares a new agent to every other agent to update the ideal distance, which is O(n), with n being the number of agents in use. Later, it compares every agent to every other agent to update the agent's velocity based on its neighboring agents. Although some suggestions for reducing the complexity such as using a neighborhood matrix, were given [117], the worst case time complexity remains $O(n^2)$, where n is the number of agents

TABLE 24. Datasets.

Dataset ID	Number of Items	Number of Attributes	Number of Clusters	Maximum Similarity	Average Similarity
I	1600	2	2	0.999948	0.814345
II	811	2	3	0.997500	0.732619
Iris	150	4	3	1.0	0.709334
Movielens	10,000 ratings 943 users	1682 movies	NA	1.00	0.23

in use. Similarly, the memory complexity is also $O(n^2)$ to keep the ideal distances, in addition to $O(n)$ memory needed for keeping the agent locations, velocities and amplitudes.

5.4 Experiments in Dynamic Clustering

In this section, we describe our datasets, experiments, and evaluation metrics; and discuss our results.

5.4.1 Datasets

Table 24 presents the datasets used in Dynamic-FClust experiments. Dataset I and Iris are the same datasets that were used in the previous chapters. As a reminder, dataset I [2] is a synthetic dataset, whereas Iris is a real life dataset from the UCI machine learning repository [3].

The third dataset used in the experiments is the MovieLens dataset[4], which is also a real life data.

Datasets I has 2 attributes and is thus suitable to show the emerging and disappearing clustering results visually. In the literature, FClust has been compared to other algorithms and one of the datasets used was the Iris dataset. Thus, to compare the Dynamic-FClust Algorithm with the original FClust algorithm, the Iris dataset is also used. Finally, the Movielens dataset is a dataset with dynamic data items. At each time step, one rating in the format of $< userid, movieid, rating, time >$ is provided to the dynamic-FClust system.

In the experiments, the Manhattan based similarity, given in Equation (4) is used for dataset I and for the *linearly normalized* Iris dataset. For the MovieLens, the Pearson correlation, in Equation (13), is used.

[2] http://webmining.spd.louisville.edu/NSF_Career/datasets.htm
[3] http://archive.ics.uci.edu/ml/
[4] http://www.grouplens.org/node/73

5.4.2 Clustering Evaluation

To evaluate and compare the quality of clustering results the average inter-cluster and intra-cluster similarities are computed. The *inter-cluster* similarity is the similarity between clusters and the *intra-cluster* similarity is the similarity between data points within a cluster. A higher intra-cluster similarity and lower inter-cluster similarity represent better clustering quality. More information is presented in Section 2.1.4.

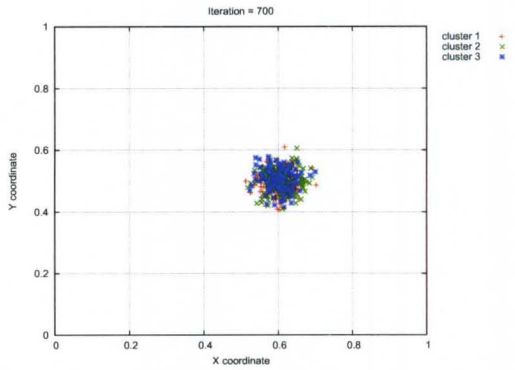
Additionally, when there are class labels available, a cluster error may be computed by comparing the cluster and class labels. The procedure given in Algorithm 3 checks whether each item pair has the same class label if they have the same cluster labels or not.

5.4.3 Results

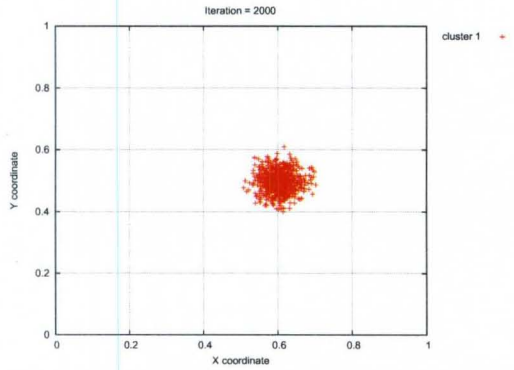
Our experimental results show that Dynamic-FClust is capable of handling emerging and disappearing clusters, as well as dynamic data items, while producing similar clustering and visualization results to FClust in terms of the number of clusters extracted, cluster error, inter-, and intra-cluster similarity.

In the figures, N_MIN represents the minimum cardinality, which is the minimum number of data records required for a cluster to be considered valid.

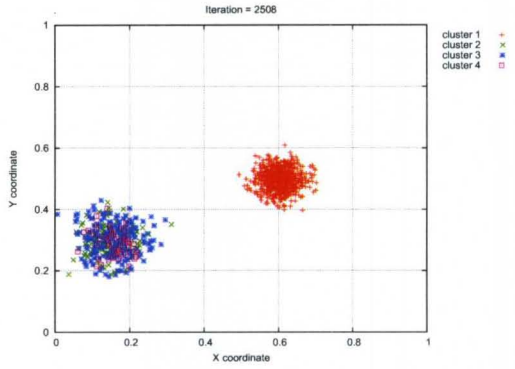
Figure 33 shows the process of clustering a dataset with two clusters using Dynamic-FClust vs. FClust. This experiment also illustrates the ability to capture emerging clusters using Dynamic-FClust. Initially, the dataset consisted of only one cluster. Figure 33(a) shows that, before convergence, Dynamic-FClust extracted 3 data clusters. Around 2000 iterations, Dynamic-FClust converged as shown in Figure 33(b). Around 2500 iterations, new data items from a new cluster emerged. Without re-starting the clustering process, Dynamic-FClust captured the emerging cluster successfully, as shown in Figure 33(d). Figures 33(e) and 33(f) visually show that Dynamic-FClust produced similar clustering and visualization results to FClust, although initially, the dataset was not fully available for dynamic clustering. Additionally, in some cases, FClust may take longer to converge than Dynamic-FClust.



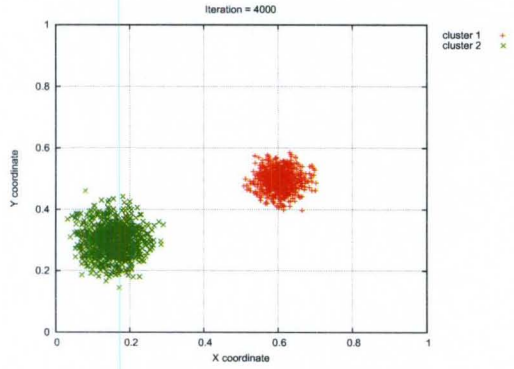
(a) Clustered dataset after first cluster was read in Dynamic-FClust (iteration=700).



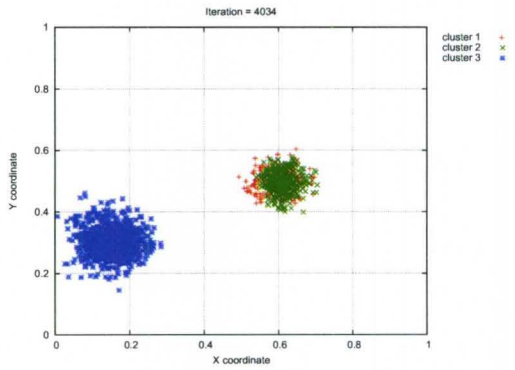
(b) Clustered dataset after first cluster was correctly clustered using Dynamic-FClust (iteration=2000).



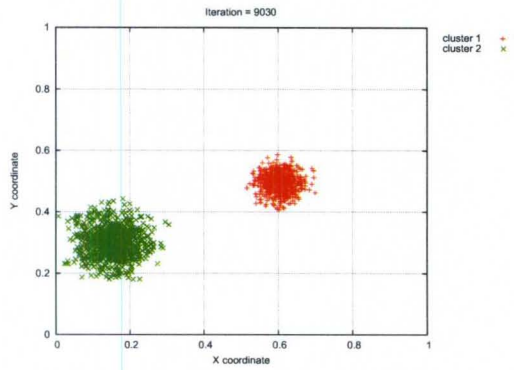
(c) Clustered dataset. Emerging cluster being processed using Dynamic-FClust(iteration=2500).



(d) Clustered dataset. Emerging cluster discovered, clustering process converged using Dynamic-FClust(iteration=4000).



(e) Clustering process using FClust (iteration 4034).



(f) Clustering process converged using FClust (iteration 9000).

Figure 33. Clustering a dataset with two clusters using FClust vs. Dynamic-FClust where $d_{th} = d_{ideal\_th} = 0.04$, $sim_{th} = 0.91$, $N\_MIN = 40$.

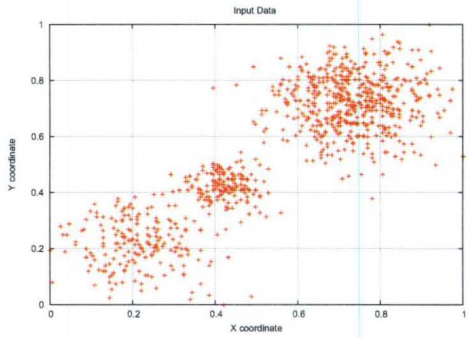
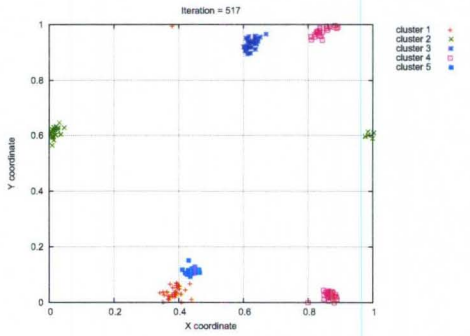
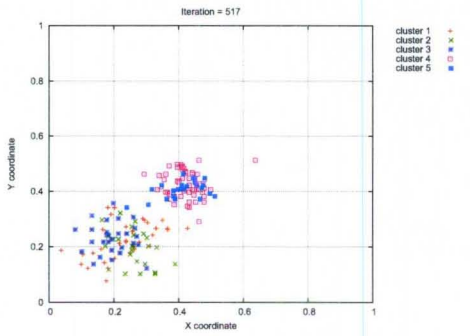


Figure 34. The 3 cluster dataset.

Figure 34 shows the 3 cluster dataset that was also used in experiments in Chapter 3.



(a) Agent clusters on the visualization screen illustrating the wrap-around cluster formation.

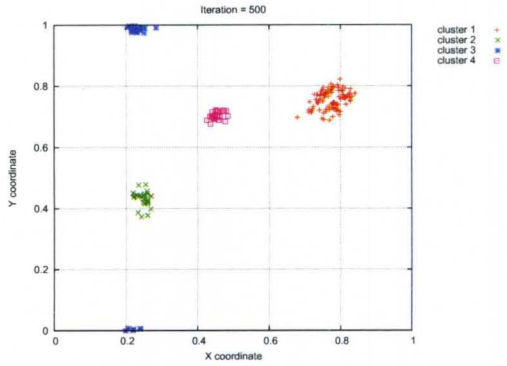


(b) Data clusters visualized to illustrate the wrap-around cluster formation result.

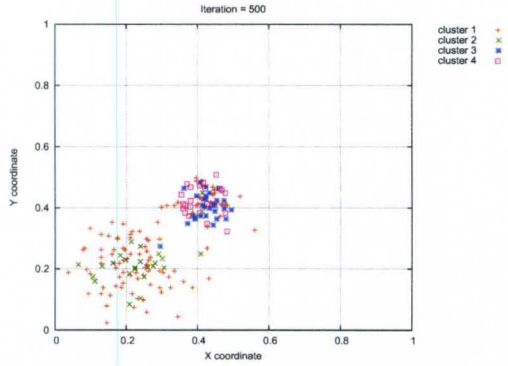
Figure 35. Wrap-around cluster formation illustrated, the 3-cluster dataset is being clustered using Dynamic-FClust, where $d_{th} = d_{ideal\_th} = 0.04$, $sim_{th} = 0.91$, $N\_MIN = 20$.

Figure 35 illustrates the wrap-around cluster formation on the agent visualization panel while clustering the 3-cluster dataset using Dynamic-FClust. Having the same color on Figure 35(a) means that the agents are in the same cluster. Additionally, cluster colors for the agent domain represent the same clusters in the data domain, on Figure 35(b).

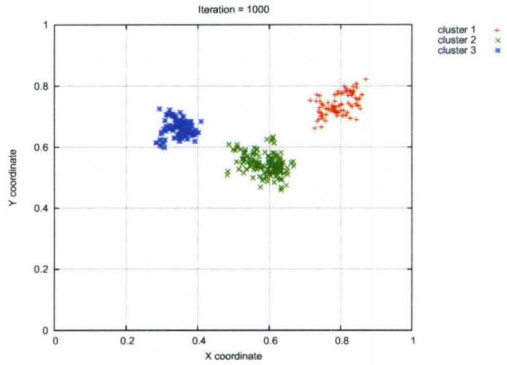
One of the capabilities of dynamic clustering is the ability to delete clusters and data records



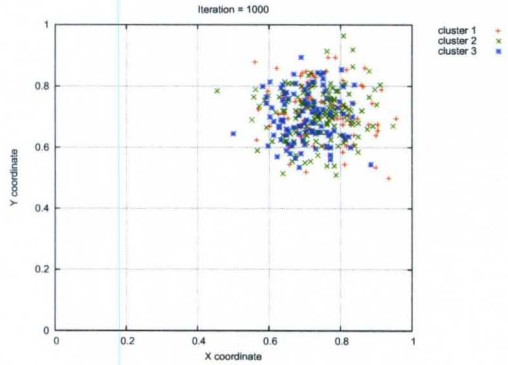
(a) Agent clusters on the visualization screen, the newest 500 data records are represented (time step=500).



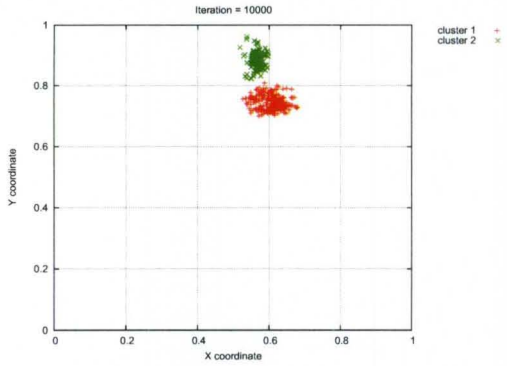
(b) Data clusters, the newest 500 data records are clustered (time step=500).



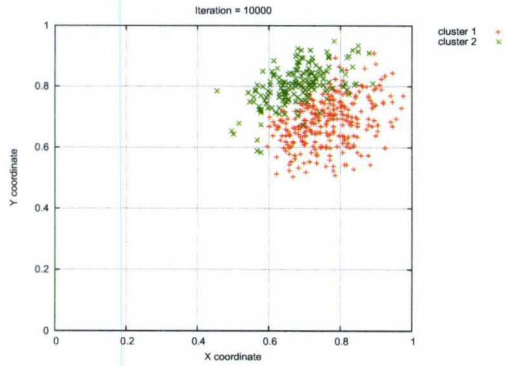
(c) Agent clusters on the visualization screen, the newest 500 data records are represented (time step=1000).



(d) Data clusters, the newest 500 data records are clustered (time step=1000).



(e) Agent clusters on the visualization screen, the newest 500 data records are represented (time step=10000).



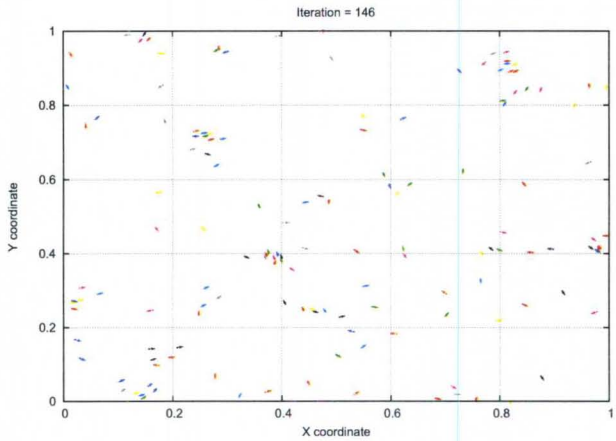
(f) Data clusters, the newest 500 data records are clustered (time step=10000).

Figure 36. Illustrating cluster deletion and keeping the newest 500 data records. The 3-cluster dataset is being clustered using Dynamic-FClust, where $d_{th} = d_{ideal\_th} = 0.04$, $sim_{th} = 0.91$, $N\_MIN = 20$.

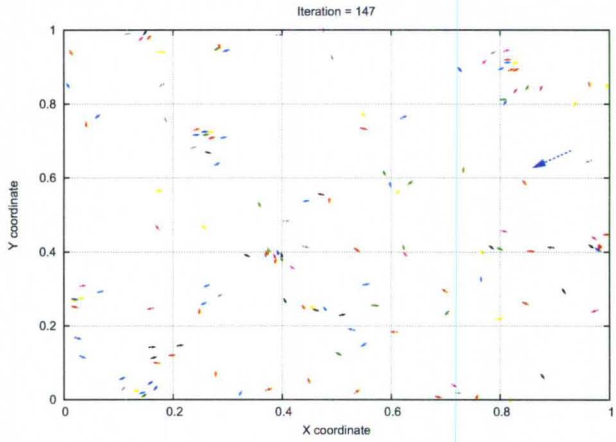
with time if desired. The data record may be expired and removed from the dataset, given in the Dynamic-FClust Algorithm, Algorithm 17, line 5, or there may be an upper bound for the number of agents due to memory or computational limitations, as presented in line 13 of Algorithm 17. Figure

36 illustrates that Dynamic-FClust is capable of handling data removal, and is able to update the clustering results during data removal. Unlike other static clustering algorithms, Dynamic-FClust does not require a re-start. In the experiments, it is assumed that the most recent (up to 500) records are wanted while the older or expired data records are to be deleted. Thus, at most 500 agents were used. It is assumed that, at each time step, one record was available. Thus, Dynamic-FClust keeps inserting the new agents in the first 500 time steps. Staring from the 501st time step, Dynamic FClust keeps the most recent 500 data records and deletes the previous data records. Figure 36(c) shows the original data colored by the cluster label assigned by Dynamic-FClust. This explains that, in the first 500 time steps, data records were from the 2 clusters of the original dataset located on the lower left corner and they are clustered into 4 clusters by Dynamic-FClust. Dynamic-FClust would need more time to achieve the final clustering results. In the mean time, until time step 812, input data feeds to the Dynamic-FClust. With the addition of each new arriving data record, the oldest data records gets removed from the dataset, accompanied by the removal of the representing agent from the visualization panel. Dynamic-FClust keeps iterating and updating the clustering results. At time step 1000, Figure 36(d) shows that the remaining data records are from the upper right cluster. With the removal of the data records from the lower left corner, the clusters that they belong to are also removed and the newest 500 data records are clustered into 3 clusters by Dynamic-FClust. As the time to improve clustering results is given to Dynamic-FClust, results continue improving and the remaining data gets clustered into 2 clusters, as shown in Figure 36(e) and Figure 36(f). Additionally, looking at the visualization panel given in Figure 36(e), we observe that the two clusters extracted are very similar and may get merged after more iterations. To sum up, Dynamic-FClust is capable of handling both data addition and data removal in time, all the while updating clusters accordingly, including deleting data clusters if needed.

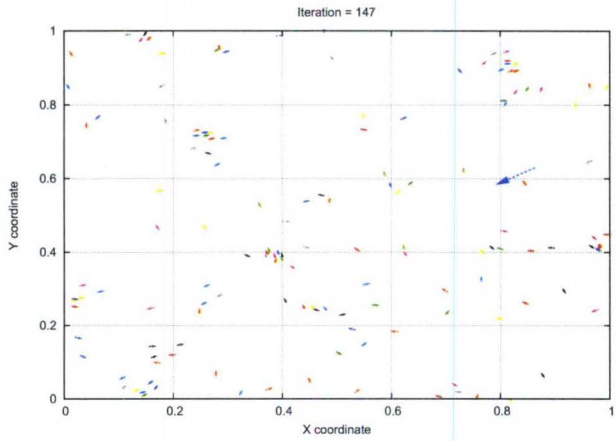
If desired, different approaches such as freezing and modified neighborhood definition can be tested to see the effects on clustering. In freezing, when there is a new data entry, all the agents except the active agent, which represents the new data entry, stops moving. The active agent updates its position and velocity using the feedback (a.k.a attractive and repelling forces) from its neighborhood, thus becoming closer to other similar agents. Another option is to have a bigger neighborhood for the active agent to provide more feedback from the neighbors. The benefits of



(a) Visualization panel before a new rating is read and after all agents updated their velocities and locations (time step=146).



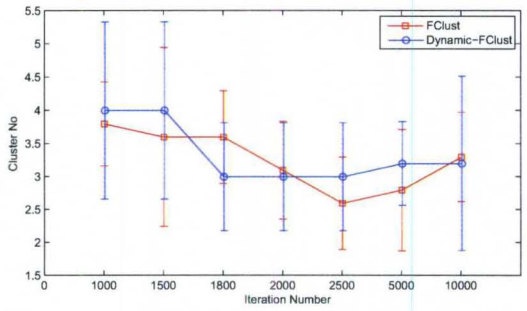
(b) Visualization panel when the new rating is read at time step=147. All other agents are frozen and the newly updated agent uses a bigger neighborhood and amplitude.



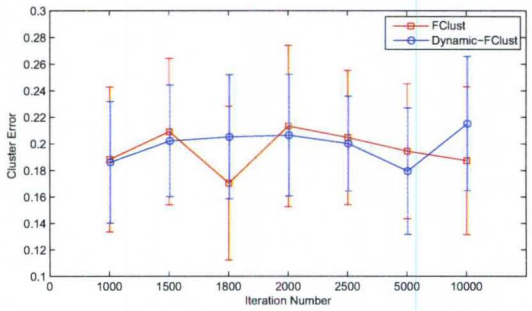
(c) Visualization panel showing that the newly updated agent keeps updating it's position and velocity while all other agents are frozen.

Figure 37. Illustrating freezing and an expanded neighborhood around a new agent. The 3-cluster dataset is being clustered using Dynamic-FClust, where $d_{th} = d_{ideal\_th} = 0.04$, $sim_{th} = 0.91$. During freezing, $d_{th} = d_{ideal\_th} = 0.4$ for the modified/unfrozen/active agent.

using an adaptive neighborhood threshold, which gets smaller in time, were presented for static clustering, as part of the FClust-Annealing algorithm in Section 3.1.2. Thus, Figure 37 illustrates the freezing and modified neighborhood for a newly updated agent. In Figure 37(a), all the agents update their positions and velocities. Later, new data arrives. All agents except the active agent froze. The blue arrow in Figure 37(b) and Figure 37(c) represents the active agent. While all the other agents are frozen, the active agent keeps moving on the visualization screen. Additionally, to provide a wider observation opportunity, the distance threshold is increased to 0.4, $d_{th} = 0.4$ for only the active agent. The arrow lengths on the visualization panel represent the amplitude of the velocity of the agents, and the amplitude of the velocity of the active agent is increased by setting $d_{ideal\_th} = 0.4$. During freezing, the worst case computational complexity is $O(n)$. After the active agent is done with updating its position and velocity, the freezing stops, Dynamic-FClust continues normally, and thus, all agents update their velocities and positions. Since our results of Dynamic-FClust are already superior to static FClust, we will not go into the details of improving the results by freezing and/or adaptive thresholding in Dynamic-FClust.

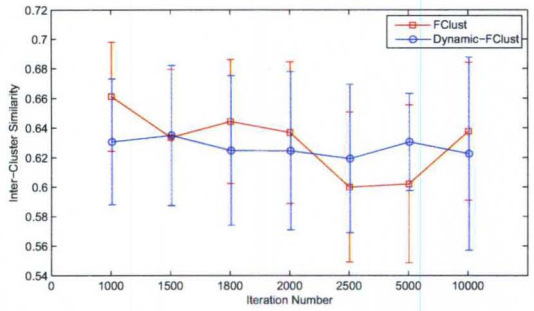


(a) Number of clusters generated via FClust vs. Dynamic-FClust. Interval marks represent standard deviation.

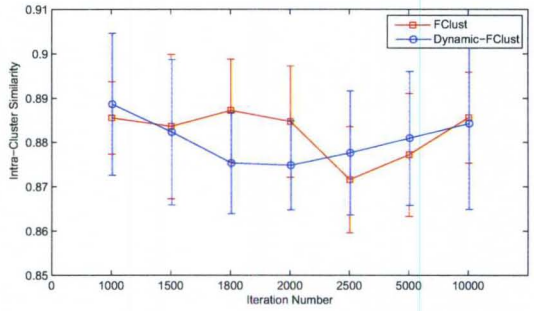


(b) Cluster error, FClust vs. Dynamic-FClust. Interval marks represent standard deviation.

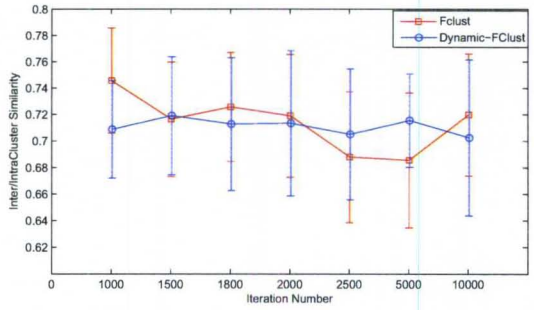
Figure 38. Average results (and standard deviations) of 10 different runs of clustering the Iris data using FClust vs. Dynamic-FClust, compared via extracted cluster number and cluster error computed using Algorithm 3 , where $d_{th} = d_{ideal\_th} = 0.04$, $sim_{th} = 0.85$, $N\_MIN = 7$.



(a) Inter-similarity of FClust vs. Dynamic FClust. Interval marks represent standard deviation.



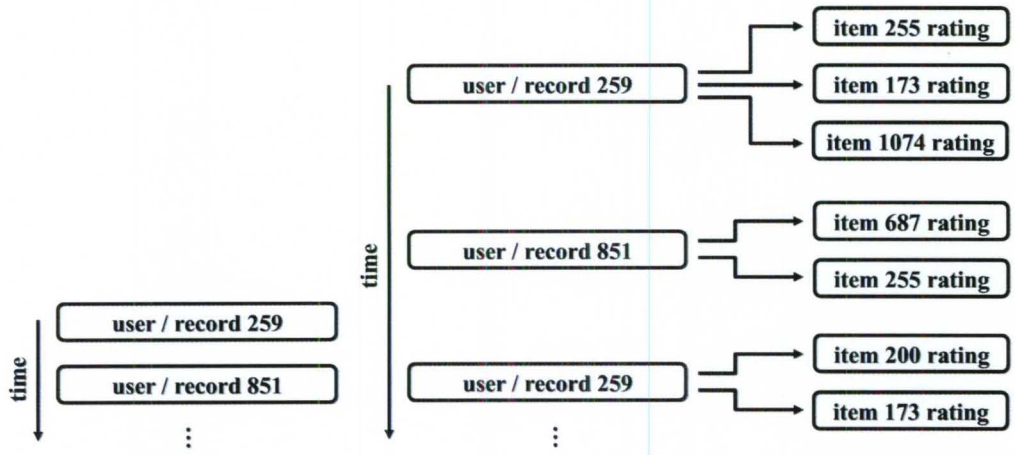
(b) Intra-similarity of FClust vs. Dynamic FClust. Interval marks represent standard deviation.



(c) The ratio of inter-similarity to intra similarity, FClust vs. Dynamic-FClust. Interval marks represent standard deviation.

Figure 39. (Average results (and standard deviations) of 10 different runs of clustering the Iris data using FClust vs. Dynamic-FClust, compared via inter- and intra-cluster similarity metrics, where $d_{th} = d_{ideal\_th} = 0.04$, $sim_{th} = 0.85$, $N\_MIN = 7$.

Figure 38 and Figure 39 compare the results of 10 different runs of clustering the Iris data using FClust and using Dynamic-FClust. The horizontal lines show the average values whereas the vertical lines represent the standard deviations. Figure 38(a) shows that Dynamic-FClust extracts a similar number of clusters as FClust, and clusters keep getting updated as long as the algorithm runs. Figure 38(b) shows that the cluster error has a similar range in both Dynamic-FClust and FClust. Due to the data availability difference, Dynamic-FClust and FClust may find similar results



(a) Incremental data input style example.

(b) Dynamic data input style example.

Figure 40. Visualization of incremental vs. dynamic data reading. (a) shows incremental and (b) shows dynamic. Note that in dynamic data, the same entry can be re-rated multiple times by the same user in any order, thus changes to the same user and item are possible over time.

in different iteration numbers. Figure 39 compares the two clustering approaches in terms of the inter- and intra-cluster similarity. In Figure 39(c), Dynamic-FClust creates better clusters initially, although the whole data was only available to Dynamic-FClust after 150 iterations. This may be because fewer data items result in a lower number of constraints. Dynamic-FClust satisfies some of the constraints better, before accepting more constraints to satisfy. Although both FClust and Dynamic-FClust act similarly in the long term, with its lower number of iterations, Dynamic-FClust may be preferred over FClust for clustering the Iris dataset. To summarize the results, Figure 38 and Figure 39 confirm the fact that Dynamic-FClust and FClust extract clusters with similar quality and that they may require a different number of iterations to find an optimal solution.

Clustering Movielens: Dataset with dynamic data records

Figure 40 presents the difference between data reading in incremental vs. dynamic reading. In dynamic data, the same entry can be re-rated multiple times by the same user in any order, thus changes to the same data record and data item are possible over time.

Figure 41 shows the number of users joining the dataset vs. the time step, where one rating is available at each iteration. All 943 users become available at iteration $99,940$ whereas the whole dataset becomes available after $100,000$ iterations.

Figure 42 presents the similarity histogram and log-log plot similarities for the MovieLens dataset. Figure fig:movielensDIdealHist presents the ideal distances measured using the similarities

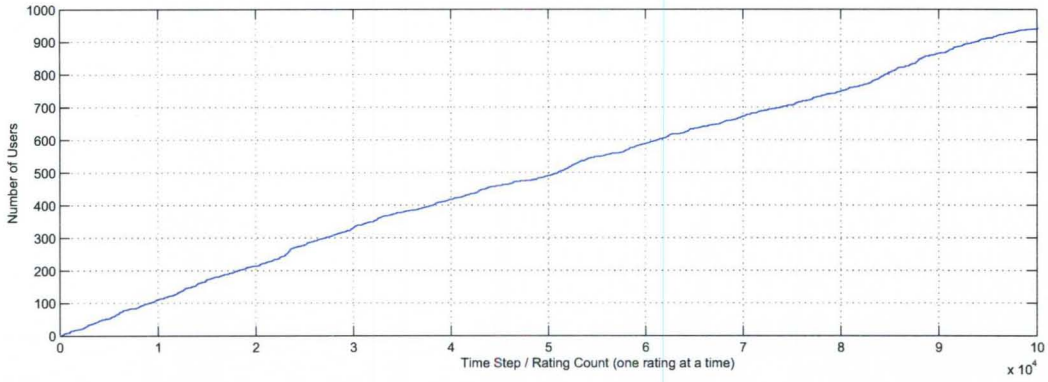


Figure 41. Number of users vs. time step vs. where dynamic-FClust reads one rating per time step.

TABLE 25

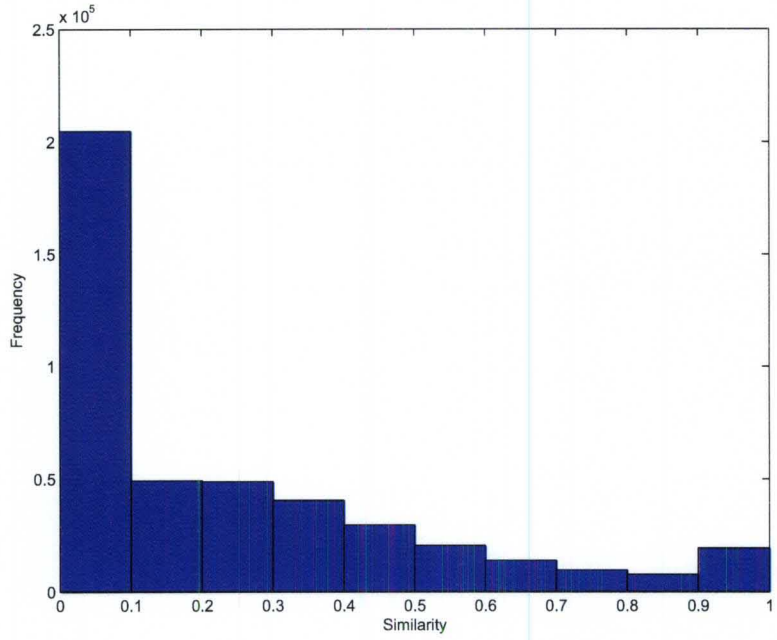
Comparison of Dynamic-FClust vs. FClust, when the whole Movielens dataset has just been read. A sample run results after post-processing, using wrap-around cluster formation, where $d_{th} = d_{ideal\_th} = 0.04$, $sim_{th} = 0.22$, $N\_MIN = 10$.

Algorithm	Cluster No	Outlier No	Inter-Sim	Intra-Sim	Inter/Intra
FClust	22	485	0.24	0.25	0.93
Dynamic-FClust	7	33	0.23	0.26	0.90

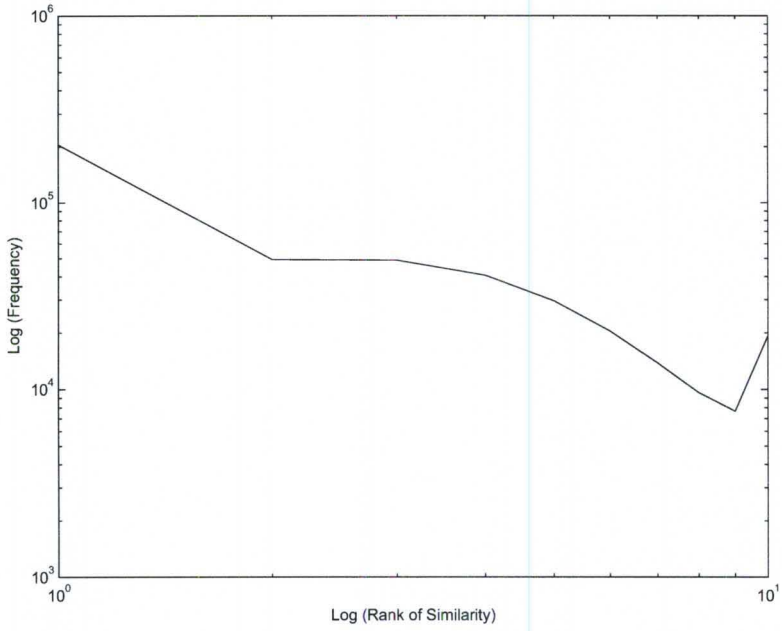
in Figure 42.

Figure 44 shows the results of clustering the Movielens dataset using Dynamic-FClust and using FClust. In the experiments, the Movilens data is sorted by rating time and at each iteration, one line of information, which is in the form of <user id, movie id, rating, rating time> is supplied to the Dynamic-FClust. It is observed that, Dynamic-FClust is capable of handling dynamic clusters and dynamic data items. At time t, when the whole dataset is available, as a static clustering algorithm, FClust has no information coming from the time step t' where $t' < t$. Whereas in Dynamic-FClust, at time t, some cluster information is already available from the previous time step t'. This example clearly shows the advantage of Dynamic-FClust over static clustering algorithms, such as of FClust.

Table 25 compares the results of Dynamic-FClust vs. FClust, when the whole Movielens dataset has just been read. This table exemplifies the advantage of Dynamic-FClust over FClust. When the whole dataset becomes available at time step $100,000$, Dynamic-FClust uses the information from previous time steps, whereas as a static clustering algorithm, FClust starts with no previous information. Clearly, results of Dynamic-FClust are better than FClust. The difference is mostly observable via the number of users that were marked as outliers. Such a big difference in



(a) Pearson similarity histogram.

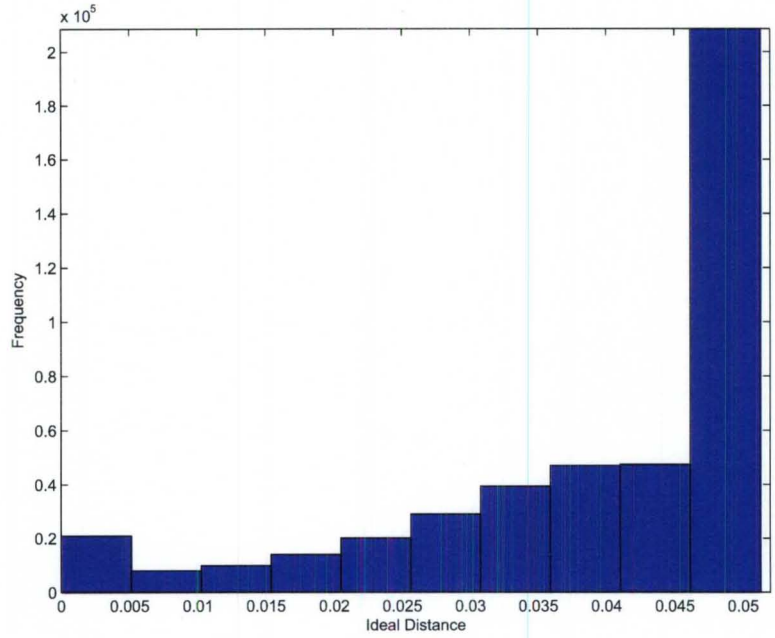


(b) Log-log plot of similarities.

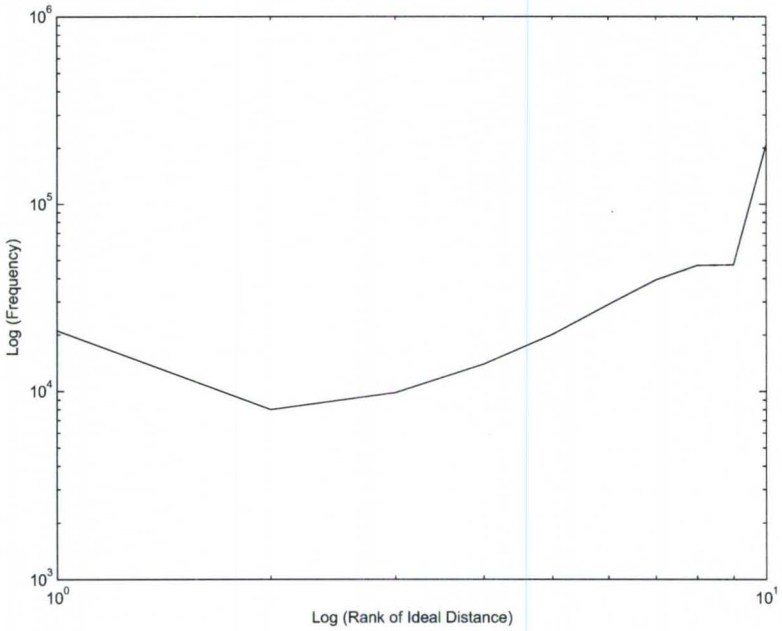
Figure 42. Pearson similarity was computed for 444, 153 user pairs for the Movielens dataset, which has 943 users. The data shows power law properties. The average similarity is 0.23.(a) Similarity histogram, (b) Log-log plot of similarities exhibiting power law properties.

outlier no shows that FClust has not converged yet.

Table 26 shows same sample profiles extracted just when the whole dataset became available. Movies with average ratings 2.00 or above are listed in the table. Looking at the profiles, we observe



(a) Ideal distance histogram.

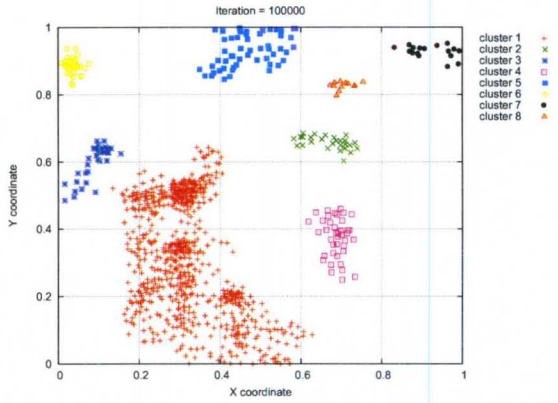


(b) Log-log plot of ideal distances.

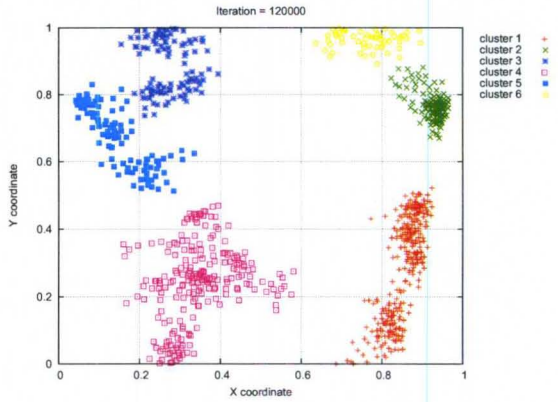
Figure 43. Ideal distances histogram for the Movielens dataset. Ideal distances are computed using Equation (28). (a) Ideal distance histogram, (b) Log-log plot of ideal distances exhibiting power law properties (linearity).

that users in cluster 5 rate more frequently and give higher ratings. It looks like Star Wars (1977) is a favorite of all profiles but some groups of people like it more than others.

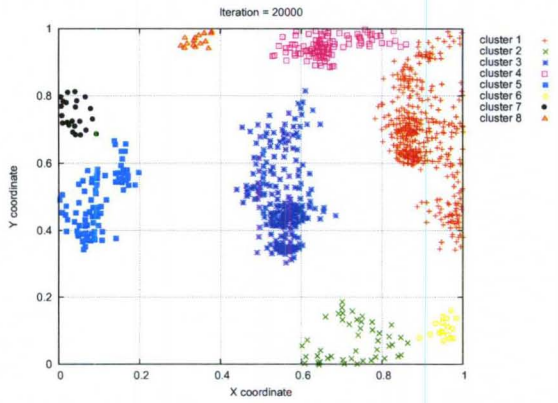
Figure 45 presents the total number of movies rated by at least one user until the corre-



(a) Dynamic-FClust, when the whole dataset has just been read.



(b) Dynamic-FClust, at 20000 iterations after the whole dataset has been read .



(c) FClust, at 20000 iterations after the whole dataset is read.

Figure 44. Clustering the Movielens dataset using Dynamic-FClust vs. FClust, visualization panel with clustered agents where $d_{th} = d_{ideal\_th} = 0.04$, $sim_{th} = 0.22$, $N\_MIN = 10$.

sponding time step for the first $10,000$ ratings. In a dynamic dataset, same movie may be rated by different users as well as it can be re-rated by the same user.

TABLE 26

Several examples from the profiles of MovieLens data extracted using Dynamic-FClust where $N\_MIN = 10$, $timestep = 100,000$. Movies with average ratings 2.00 or above are listed in the table.

Average Movie Rating	**Movie**
	Profile 1 (includes 310 users)
2.58	Star Wars (1977)
2.35	Fargo (1996)
2.15	English Patient, The (1996)
2.06	Return of the Jedi (1983)
2.00	Raiders of the Lost Ark (1981)
	Profile 2 (Includes 43 users)
2.71	Star Wars (1977)
2.22	Return of the Jedi (1983)
2.09	Contact (1997)
2.00	Fargo (1996)
	Profile 3 (Includes 40 users)
2.45	English Patient, The (1996)
2.30	Star Wars (1977)
2.08	Contact (1997)
2.03	Scream (1996)
2.00	Fargo (1996)
	Profile 5 (Includes 42 users)
3.93	Star Wars (1977)
3.40	Return of the Jedi (1983)
2.71	Fargo (1996)
2.48	Raiders of the Lost Ark (1981)
2.45	Godfather, The (1972)
2.33	Independence Day (ID4) (1996)
2.33	Contact (1997)
2.26	Toy Story (1995)
2.17	Mr. Holland's Opus (1995)
2.12	Empire Strikes Back, The (1980)
2.12	Star Trek: First Contact (1996)
2.02	Monty Python and the Holy Grail (1974)

Figure 46 presents the average number of movies rated per user vs. the time step. The user may submit ratings in consecutive time steps or there may be a time gap between ratings from the same user. The addition of new users cause vertical drops in the plot. In the MovieLens dataset, the number of movies represents the dimensionality. In high dimensional datasets, the more the user rates movies, the better. When a user rates a relatively small number of movies, the data becomes sparse. Comparing the number of movies in Figure 45 and the average number of rated movies per user in Figure 46, we observe that the MovieLens dataset is a sparse dataset, which makes clustering more challenging.

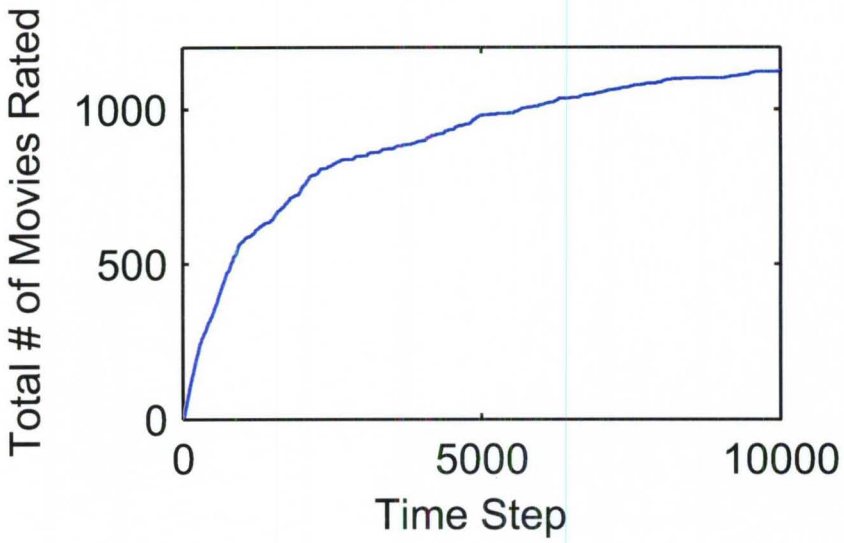


Figure 45. Total number of movies rated vs. the time step for the 10 percent of the ratings in the Movielens data set.

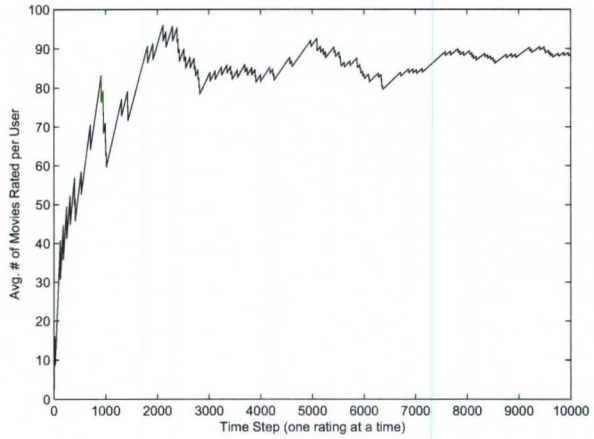


Figure 46. The average number of movies rated per user vs. the time step for the first 10 percent of the MovieLens ratings.

Figure 47 presents the average rating per user vs. the time step. In the MovieLens dataset, a higher rating means user liked the movie. Additionally, a high rating average may imply an optimist user.

Figure 48 presents the average rating per movie vs. time step. Some movies may be rated higher when they are newer and more popular. Thus, the average of ratings may decrease in time. When the movies become old and/or no longer popular, they may be considered as uninteresting by the users and they may not be rated as often.

Figure 49 presents the average results of 10 different runs of clustering 10 percent of the

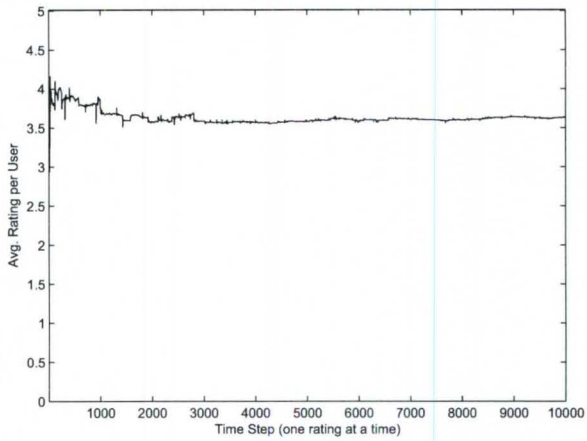


Figure 47. The average rating per user vs. the time step for the first 10 percent of the MovieLens ratings.

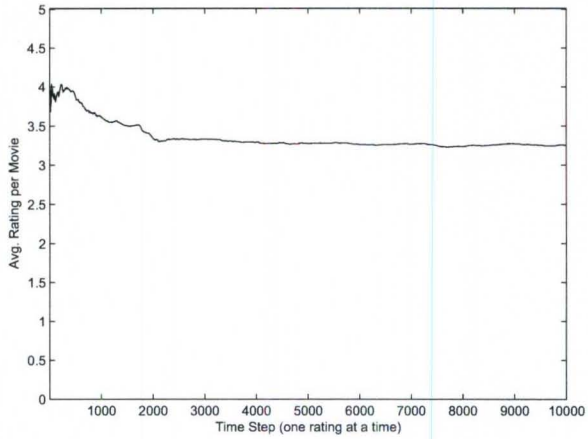
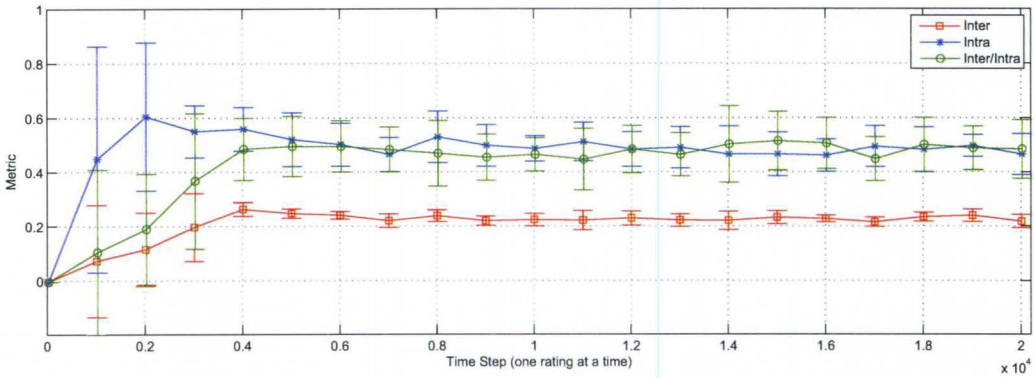


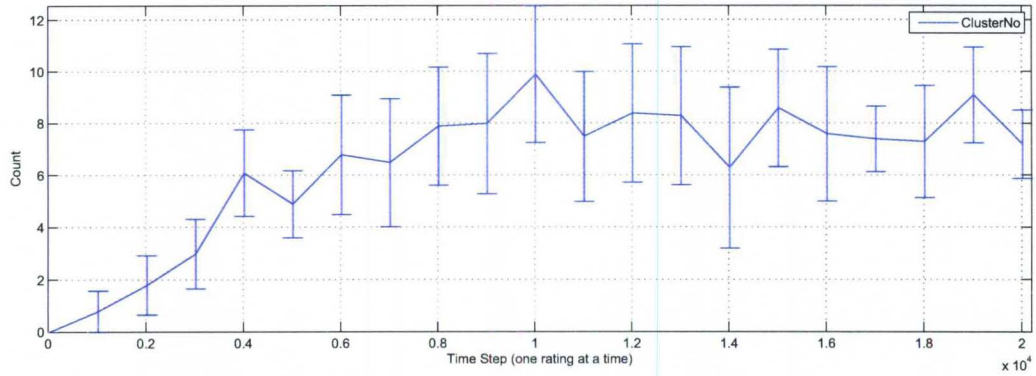
Figure 48. The average rating per movie vs. the time step for the first 10 percent of the MovieLens ratings.

MovieLens dataset using Dynamic-FClust. In the experiments, one rating becomes available to Dynamic-FClust, at each time step, for the first $10,000$ time steps. Thus, the whole dataset is available to Dynamic-FClust starting from time step $10,000$. The horizontal lines show the average values whereas the vertical lines represent the standard deviations. Figure 49(a) show that after some iterations, the clustering results stabilize.

Figure 50 illustrates the advantage of dynamic clustering over static clustering by comparing the results of Dynamic-FClust to FClust. The results are average (and standard deviations) of 3 different runs of clustering the MovieLens dataset using FClust and Dynamic-FClust after post-processing, using the wrap-around cluster formation, where $d_{th} = d_{ideal\_th} = 0.04$, $sim_{th} = 0.345$, and $N\_MIN = 10$. Vertical interval marks represent standard deviation. The results are compared via inter-similarity, intra-similarity, inter-similarity over intra-similarity, and extracted cluster num-



(a) Inter-cluster and intra-cluster similarities. Vertical interval marks represent standard deviation.



(b) Number of clusters extracted. Interval marks represent standard deviation.

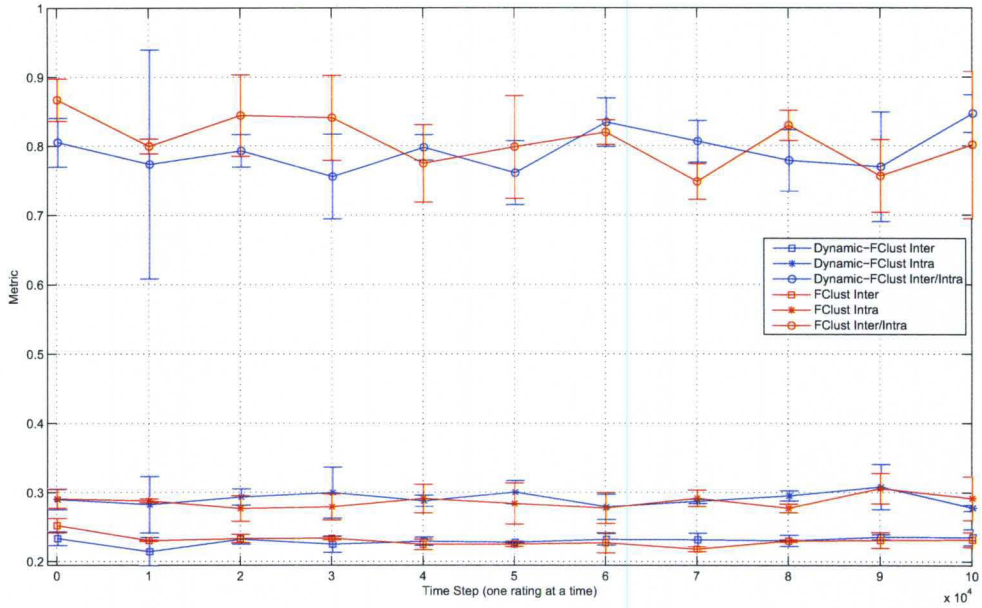
Figure 49. Average results (and standard deviations) of 10 different runs of clustering the first 10 percent of the MovieLens dataset using Dynamic-FClust after post-processing, where $d_{th} = d_{ideal\_th} = 0.04$, $sim_{th} = 0.345$, $N\_MIN = 2$.

ber. The plots show the results starting from the first time step where the entire dataset becomes available, and goes $100,000$ more time steps. Clusters were evaluated as soon as the whole dataset became available and both Dynamic-FClust and FClust updated the agents' properties, such as position and amplitude, for one iteration. There is no more data input to any of the algorithms after that time step, thus both algorithms continue updating their agent properties, i.e. improving their clustering results. Meanwhile, cluster evaluations are repeated at every $10,000th$ time step. Dynamic-FClust receives one-rating at a time, then updates its clustering results, and stores the information it was able to extract. When the $100,000th$ rating was available, Dynamic-FClust updated its agent population and continued improving its cluster formation. On the other hand, FClust started clustering from scratch when the whole dataset became available, thus, it took approximately $20,000$ time steps for FClust to produce similar results to Dynamic-FClust. These results illustrate the effectiveness of using the information from the past and advantages of Dynamic-FClust over

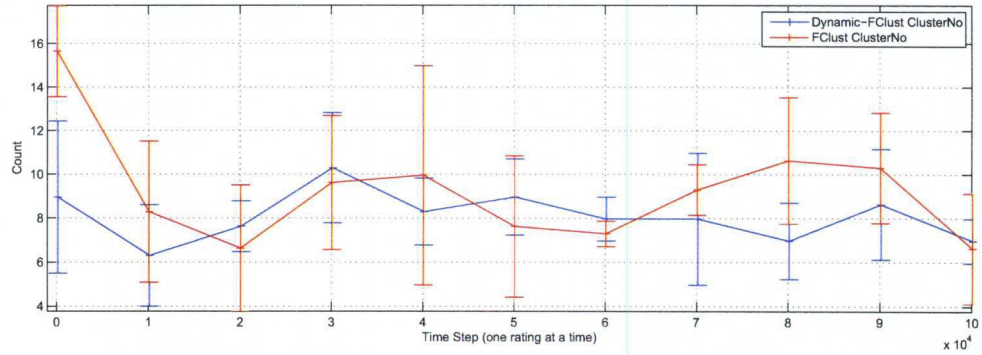
FClust.

5.5 Conclusions

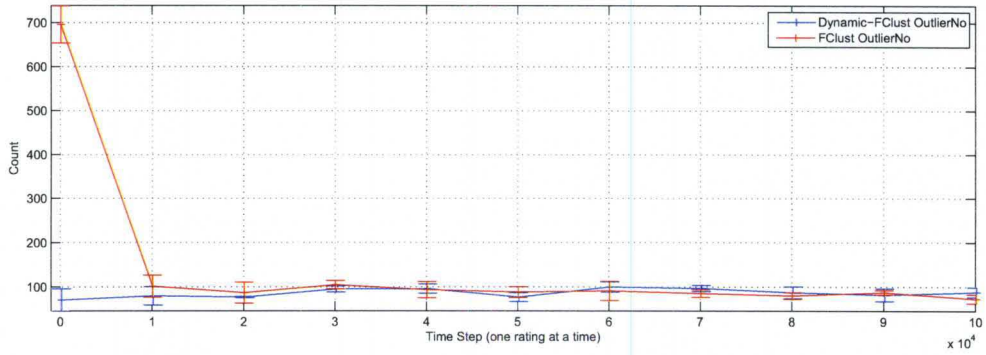
Our previous studies on various synthetic and real life datasets showed that algorithms that are based on flocks of agents can cluster and visualize the data simultaneously [128, 129]. In this chapter, we developed an algorithm for dynamic clustering and visualization, which is capable of handling dynamic data items as well. We observed that, similar to FClust, in Dynamic-FClust parameter setting plays a crucial role in convergence. FClust has quadratic complexity. Our experimental results showed that Dynamic-FClust can cluster and visualize dynamic data simultaneously, while capturing emerging clusters and producing clustering and visualization results that are comparable to FClust.



(a) Inter-cluster and intra-cluster similarities using FClust and Dynamic-FClust.



(b) Number of clusters extracted using FClust and Dynamic-FClust.



(c) Number of outliers extracted, FClust vs. Dynamic-FClust.

Figure 50. Average results of clustering the MovieLens dataset using FClust and Dynamic-FClust. Vertical interval marks represent standard deviation.

CHAPTER 6

CONCLUSIONS AND FUTURE DIRECTIONS

The natural analogy between Swarm Intelligence (SI) systems and social behavior have been the main motivation for the topic of investigation in this dissertation, with a focus on Flock based systems which are a particularly attractive SI systems that possess unique properties for modeling dynamic online social activity, and yet have not been well investigated for this purpose. More specifically, we investigate the use of flock-based swarm intelligence for two related and challenging problems that form critical building blocks of intelligent personalized web servers, specifically

(i) the problem of understanding the online activities of online users by discovering groups or clusters of similar users, and

(ii) the problem of predicting the interests of online users in anticipation of further decision making goals, in particular focusing on collaborative filtering or social recommender systems.

The most popular SI clustering techniques are Particle swarm clustering and Ant-based clustering. More recently, clustering using flocks of agents proved to be promising as well. Since in this approach, each agent represents a data item, and the distance between agents on the visualization panel depends on the similarity between the data records represented by these agents, the flocks-of-agents approach, known as FClust, offers a solution not only for clustering but also for data visualization. Additionally, FClust possesses a dynamic structure. Agents keep moving on the visualization panel until the algorithm is forced to stop. After a sufficient number of iterations, every state of the visualization panel may provide a clustering alternative for the dataset. This makes FClust suitable for clustering and visualization of dynamic datasets. Although the initial experimental results with FClust were acceptable, it suffered from several limitations, namely lack of scalability, and inadequacy to for clustering high dimensional sparse dataset, in particular dynamic transactional data.

To overcome some of the observed limitations , we have proposed some improvements including FClust-annealing, the (K-means+FClust) Hybrid, and the (SPKM+ FClust) Hybrid algorithms. Annealing decreased the number of iterations to convergence and improved the quality of

the clusters. The effect of different cooling functions should be studied further. In addition to these improvements, the proposed (K-means+FClust) hybrid algorithm reduces the quadratic complexity of FClust to linear complexity and performs similarly to FClust with fewer iterations during the experiments.

Our work on Web server access log data, a special case of high-dimensional and sparse data, showed that algorithms that are based on flocks of agents can cluster and visualize the data simultaneously [128, 129]. Our (SPKM+FClust) hybrid approach reduces the quadratic complexity of FClust to linear complexity, and performs similarly to FClust, but has the advantage of fewer iterations for clustering large, high-dimensional data such as web usage data. For Web usage data, our experiments confirmed the superiority of the proposed hybrid approach, both in terms of quality of the final results and computational cost.

We have also developed a new recommender system approach called the flocks-of-agents based recommender system (FlockRecom). This new approach is based on swarm intelligence, specifically, the dynamic collaboration between bird flocks in nature. The results were compared to the traditional user-based nearest neighbor collaborative filtering and FlockRecom was more successful at providing variety in the recommendations without losing recommendation quality (in terms of precision and recall). One problem suffered by some recommender systems is over-specialization. When the recommendations are limited to the user's behavior or user's profile, the user can be restricted to seeing only similar items, and there will be no randomness. In artificial intelligence, this problem is known as the exploration/exploitation dilemma. Although collaborative filtering can counteract over-specialization by suggesting different items, the dynamic structure of the FlockRecom algorithm makes it more successful at solving the exploration/exploitation dilemma, which is also practically observed in the experimental results. This and the dynamic nature of the algorithm suggests that the proposed nature inspired recommendation system looks very promising for dynamic environments, characterized by change in the data and its underlying concepts.

Finally, we introduced a new clustering paradigm: *Dynamic Data Clustering* (or dynamic clustering in short) and proposed a flocks-of-agents based dynamic clustering algorithm, Dynamic-FClust. In this new approach, not only clusters but data records themselves may change in time. For example, if the data records are users from a movie rental website, user records change as users update their reviews for the movies. In Dynamic-FClust, clusters are updated as more information becomes available. Unlike static clustering algorithms, the clustering process is not started from scratch. Rather, at time t, Dynamic-FClust uses the information coming from time $t-1$ to update

its knowledge base. Our experiments on synthetic and real life datasets have presented Dynamic-FClust's advantages over static FClust.

6.1 Summary of Contributions

Our main contributions can be summarized as follows:

- New (K-means+FClust) and (SPKM+FClust) hybrid algorithms were developed. Our hybrid approach reduces the quadratic complexity of FClust to linear complexity, and performs similarly to FClust, but has the advantage of fewer iterations for clustering large, high-dimensional data such as web usage data. Hybrid algorithms were tested on several datasets including synthetic data, real data from UCI machine learning repository, and real Web server logs and our experiments confirmed their superiority, both in terms of quality of the final results and computational costs [128, 129].

- A dynamic clustering and visualization approach that can perform dynamic clustering was developed and tested. This approach can handle the arrival of not only one data record at a time, but also "part" of a data record, such as one attribute at a time, in any order for one data record. It can also handle the modification or updating of an individual attribute (such as one item's rating) from a record and even the removal of a data record. To our knowledge, no existing clustering technique handles "partial" data addition, modification, or removal in one record.

- A new recommender system approach called the flocks-of-agents based recommender system (FlockRecom) was developed. The results were compared to the traditional user-based nearest neighbor collaborative filtering and FlockRecom was more successful at providing variety in the recommendations without losing recommendation quality [130].

- Under special simplifications we formulate the flocking behavior as a Gradient Descent optimization that minimizes a criterion that agrees with the observed behavior in flocking in the case of attraction, namely grouping (agents getting closer to each other) and alignment. In addition it is minimizing the error between ideal and agent distance values, therefore seeking a visualization of the original data onto the 2D panel that is as faithful as possible. The mathematical derivations are included under the Future Visions in the Conclusions Chapter. According to our knowledge, this a mathematical interpretation of flocking behavior that has not been accomplished so far. And we plan to pursue it further in the future.

6.2 Limitations of This Work

- As is common with all clustering techniques, our algorithms require fixing several parameters before being applied, and finding the best parameter configuration often requires trial and error for each new dataset.

- Our hybrid approach approximates the entire agent population by using a smaller population extracted by fast partitional clustering of the agent population in a preliminary phase. This allows an agent to represent more than one data record, thus increases scalability. However, the quality of this preliminary clustering is expected to depend on the number of clusters and the parameters. Also, it would be challenging to adapt the hybrid approach to a dynamic setting because the preliminary phases of partitional clustering would have to be adapted to a dynamic setting, which would make it part of the dynamic clustering and no longer a separate preliminary phase.

- The requirement to compute all pairwise similarities remains a heavy burden on the non-hybrid approaches, due to the quadratic computational and memory complexity involved.

6.3 Vision for the Future

Unlike ant-based and particle-swarm based web mining methods, flocks-of-agents based solutions are less studied. Our research should open the doors for more studies based on flocks-of-agents based methods. In particular, our research can be expanded along the following directions:

1. **Visualization:** In the dissertation, the visualization part of FClust was implemented using C, C++, and Gnuplot. However, using Gnuplot did not provide enough interaction with the user. We have started improving the visualization using C++, OpenGL, and QT and initial results are promising in that we can produce a video of the agent movement in time, with different colors assigned to the agents depending on their clusters.

2. **Cluster Evolution Tracking:** The Dynamic-FClust presents a suitable platform for tracking cluster evolution. This can be done qualitatively by observing cluster behavior on the visualization panel, and quantitatively by defining new metrics that measure changes in clusters, such as increase in size, merging between two clusters, or a split of one cluster into several clusters.

3. **Dynamic Recommender System:** We proposed a recommender system using an idea similar to static FClust. In the future, we plan to work on implementing a dynamic recommender system using an approach similar to Dynamic-FClust. In the experiments, the KDD Cup 2011 dataset will be considered. The challenge will be recommending Music Items based on the Yahoo! Music Dataset[1].

4. **Convergence Proof:** Previous research shows that flock-of-agents-based clustering converges given suitable parameters and enough iterations. One desired goal would be to mathematically prove the convergence of the agent population to a stable state. One way to accomplish this is to rewrite and combine the agent i's movement steps from our algorithms into one position update equation. In fact, it is possible to show that the change in the previous position x_i^{t-1} of agent a_i at time step t is given by:

$$\Delta x_i^t = \left[K + \frac{d_{ideal\_th}}{(\sigma|N_i|+1)}\right] \frac{\sum_{a_j \in N_i} \left(v_j^{t-1} + f\left(\Delta D_{i,j}^{t-1}, d_{th}\right)\right)\left(x_j^{t-1} - x_i^{t-1}\right)}{\left\| \sum_{a_j \in N_i} \left(v_j^{t-1} + f\left(\Delta D_{i,j}^{t-1}, d_{th}\right)\right)\left(x_j^{t-1} - x_i^{t-1}\right) \right\|}$$

where K, α, d_{ideal_th}, and d_{th} are constants that depend on the algorithm parameters, N_i is the set of neighboring agents to agent a_i, v_j^{t-1} and x_j^{t-1} are the velocity and position of agent a_j at the previous time step, respectively, and f is either an attraction or repulsion function depending on the sign of $\Delta D_{i,j}^{t-1} = d^{t-1}(i,j) - d_{ideal}(i,j)$.

5. **Formulating Flocking as an Optimization Problem:**

Given that the position of the i^{th} agent of the flock is

$$\vec{x}_i^t = \vec{x}_i^{t-1} + a_i^t \times \vec{v}_i^t \tag{39}$$

where

$$a_i^t = a_{default} + \frac{d_{th}}{20 \times (neighbor\_no(i)+1)} \tag{40}$$

$$a_i^t = a_{default} + \frac{d_{th}}{20 \times (|N_i|+1)} \tag{41}$$

and $N_i = \{j \mid d(j,i) \leq d_{th} \quad \& \quad i \neq j\}$

[1] http://kddcup.yahoo.com/datasets.php (Accessed April 27, 2011)

Let $A = K + \frac{d_{th}}{\alpha \times (|N_i^t|+1)}$

Then taking into account only the attraction effects from agent i's neighbors, we can write the change in the position of this agent as

$$\begin{aligned} \Delta\vec{x}_i^{t+1} &= \vec{x}_i^t - \vec{x}_i^{t-1} \\ &= A \times \frac{\sum_{j \in N_i^t} \left(\vec{v}_j^t + 4 \times \left(\frac{\Delta D_{ij}^t}{d_{th} - d_{ideal}(i,j)} \right)^2 \times (\vec{x}_j^t - \vec{x}_i^t) \right)}{\left\| \vec{F}_{att} \right\|} \end{aligned} \tag{42}$$

where

$$\Delta D_{ij}^t = d^t(i,j) - d_{ideal}(i,j) \text{ and } \vec{F}_{att} = \sum_{j \in N_i^t} \left(\vec{v}_j^t + 4 \times \left(\frac{\Delta D_{ij}^t}{d_{th} - d_{ideal}(i,j)} \right)^2 \times (\vec{x}_j^t - \vec{x}_i^t) \right)$$

If we assume that $\left\| \vec{F}_{att} \right\|$ is close to 1 or does not change much from one iteration to the next (i.e. it is approximately constant), then we can rewrite the position change as an update equation corresponding to one iteration of Gradient Descent for a cost function, as follows

$$\Delta\vec{x}_i^t = -\eta \frac{\delta Cost}{\delta \vec{x}_i^t} \tag{43}$$

$$\frac{\delta Cost}{\delta \vec{x}_i^t} = -\left(\sum_{j \in N_i^t} \vec{v}_j^t + \frac{4 \times \left(\Delta D_{ij}^t\right)^2}{\left(d_{th} - d_{ideal}(i,j)\right)^2} \times (\vec{x}_j^t - \vec{x}_i^t) \right) \tag{44}$$

It can be verified that one cost function that can result in the above gradient is the one minimized in the following optimization problem:

$$Minimize \left\{ \sum_{j \in N_i^t} -\vec{v}_j^t \bullet \vec{x}_i^t + \frac{2 \times \left(\Delta D_{ij}^t\right)^2}{\left(d_{th} - d_{ideal}(i,j)\right)^2} \times \left\| \vec{x}_j^t - \vec{x}_i^t \right\|^2 \right\} \tag{45}$$

In other words, the update equations, when taking only the attraction effects in account, amount to minimizing the total errors between the visualization panel distance values between agent i and its neighbors j and the ideal distance between the original data points , i.e., ΔD_{ij} as well as minimizing the distances between the ith agent and its neighbors on the visualization panel $\left\| \vec{x}_j^t - \vec{x}_i^t \right\|^2$, while also maximizing $\sum_{j \in N_i^t} \vec{v}_j^t \bullet \vec{x}_i^t$, which is the dot product between the

ith agent's position $\vec{x}_i^t$ and the mean of all its neighbors' velocities $\sum_{j \in N_i^t} \vec{v}_j^t$. Maximization of the dot product between two vectors is equivalent to maximizing the cosine of the angle between two vectors, therefore resulting in minimizing the angle between those two vectors, in other words, seeking the best alignment. This optimization criterion agrees with the observed behavior in flocking in the case of attraction, namely grouping (agents getting closer to each other) and alignment. In addition it is minimizing the error between ideal and agent distance values, therefore seeking a visualization of the original data onto the 2D panel that is as faithful as possible. Reformulation for the case of repulsion can follow similar steps, generally leading to dispersing the agents away from each other (because of the negation in the velocity).

6. **Distributed Computing:** Although complexity was not the main goal of this work, the distributed nature of the swarm intelligence paradigm holds obvious potential for scalability and robustness with cloud-based implementations. One way to achieve this goal is to distribute the data on multiple nodes, perform swarm intelligence based clustering locally, and then combine the results. This would require a judicious partitioning of the dataset. Another direction would be to distribute the operations of the flock-based clustering even if it is only for a limited number of iterations on multiple processors. The main challenge would be managing the communication between nodes in such a way as to limit communication overhead.

REFERENCES

[1] http://www.w3.org/2001/sw/.

[2] Jester dataset, http://eigentaste.berkeley.edu/dataset/.

[3] Kdd cup 1999 data, http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.

[4] Movielens dataset, http://www.grouplens.org/node/73.

[5] Uci machine learning repository, http://archive.ics.uci.edu/ml/.

[6] Ajith Abraham, Swagatam Das, and Sandip Roy. *Soft Computing for Knowledge Discovery and Data Mining*, chapter Swarm Intelligence Algorithms for Data Clustering, pages 279–313. Springer US, 2008.

[7] A. J. Abrantes and J. S. Marques. A method for dynamic clustering of data. In *Proceedings of the 9 thBritish Conference*, 1998.

[8] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions): , 2005. *IEEE Trans. Knowl. Data Eng.*, 17(6), 2005.

[9] Charu C. Aggarwal, Jiawei Han, Jianyong Wang, and Philip S. Yu. A framework for clustering evolving data streams. In *Proceedings of the 29th international conference on Very large data bases - Volume 29*, VLDB '2003, pages 81–92. VLDB Endowment, 2003.

[10] Robert Armstrong, Dayne Freitag, Thorsten Joachims, and Tom Mitchell. Webwatcher: A learning apprentice for the world wide web. In *AAAI Spring Symposium on Information Gathering*, pages 6–12. AAAI Press, 1995.

[11] H. Azzag, N. Monmarche, M. Slimane, and G. Venturini. Anttree: a new model for clustering with artificial ants. *Evolutionary Computation, 2003. CEC '03. The 2003 Congress on*, 4:2642–2647 Vol.4, Dec. 2003.

[12] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval.* ACM Press / Addison-Wesley, 1999.

[13] Marko Balabanović. An adaptive web page recommendation service. In *AGENTS '97: Proceedings of the first international conference on Autonomous agents*, pages 378–385, New York, NY, USA, 1997. ACM.

[14] Marko Balabanović and Yoav Shoham. Fab: content-based, collaborative recommendation. *Commun. ACM*, 40(3):66–72, 1997.

[15] P.F. Baldi, P. Frasconi, and P. Smyth. *Modeling the Internet and the Web. Probabilistic Methods and Algorithms.* Wiley, May 2003.

[16] C. Basu, H. Hirsh, and W. Cohen. Recommendation as classification: Using social and content-based information in recommendation. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pages 714–720, 1998.

[17] J. Becher and R. Kohavi. Tutorial on e-commerce and clickstream mining. In *First SIAM International Conference on Data Mining*, Apr 5 2001.

[18] R. Sharma H. Kaur Bedi, P. Recommender system based on collaborative behavior of ants. *Journal of Artificial Intelligence*, 2:40–55, 2009.

[19] R. K. Belew. *Finding out about: a cognitive perspective on search engine technology and the WWW.* Cambridge University Press, New York, NY, USA, 2000.

[20] R.K. Belew. Adaptive information retrieval: using a connectionist representation to retrieve and learn about documents. In *Proceedings of the 12th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 11 – 20, Cambridge, Massachusetts, United States, 1989.

[21] K. Benabdeslem and Y. Bennani. An incremental som for web navigation patterns clustering. In *26th International Conference on Information Technology Interfaces Proceedings*, pages 209–213 Vol.1, June 2004.

[22] Pavel Berkhin. Survey of clustering data mining techniques. Technical report, 2002.

[23] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, 2001.

[24] D. Billsus and M. Pazzani. Learning collaborative information filters. In *Proc. 15th International Conf. on Machine Learning*, pages 46–54, San Francisco, CA, 1998. Morgan Kaufmann.

[25] Ingwer Borg and Patrick Groenen. *Modern multidimensional scaling : theory and applications.* New York ; London : Springer, 2005.

[26] Richard Brath. Concept demonstration: Metrics for effective information visualization. In *Proceedings For IEEE Symposium On Information Visualization*, pages 108–111, IEEE Service Center, Phoenix, AZ, 1997.

[27] W. Buntine. Learning classification trees. *Statistics and Computing*, 2(2):63–73, June 1992.

[28] R. Burke. Hybrid recommmender systems: Survey and experiments. *In User Modeling and User-Adapted Interaction*, 12(4):331–370, 2002.

[29] S. Chakrabarti. *Mining the Web: Discovering Knowledge from Hypertext Data.* Morgan-Kaufmann Publishers,, 2003.

[30] Moses Charikar, Chandra Chekuri, Tomás Feder, and Rajeev Motwani. Incremental clustering and dynamic information retrieval. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, STOC '97, pages 626–635, New York, NY, USA, 1997. ACM.

[31] Yixin Chen and Li Tu. Density-based clustering for real-time stream data. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'07)*, pages 133 – 142, 2007.

[32] Zhuo Chen and Qing-Chun Meng. An incremental clustering algorithm based on swarm intelligence theory. *Machine Learning and Cybernetics, 2004. Proceedings of 2004 International Conference on*, 3:1768–1772 vol.3, Aug. 2004.

[33] Iain D. Couzin, J. E. N. S. Krause, Richard James, Graeme D. Ruxton, and Nigel R. Franks. Collective memory and spatial sorting in animal groups. *Journal of Theoretical Biology*, 218(1):1–11, September 2002.

[34] Trevor F. Cox and Michael A.A. Cox. *Multidimensional scaling.* Boca Raton, Fla. : Chapman & Hall/CRC, 2001.

[35] X. Cui and T. E. Potok. Document clustering analysis based on hybrid pso+kmeans algorithm. *Journal of Computer Sciences (Special Issue)*, Special Issue:27–33, 2005.

[36] X. Cui, T. E. Potok, and P. Palathingal. Document clustering using particle swarm optimization. In *IEEE Swarm Intelligence Symposium*, 2005.

[37] Xiaohui Cui and Thomas E. Potok. A distributed agent implementation of multiple species flocking model for document partitioning clustering. In *Cooperative Information Agents X (Lecture Notes in Computer Science)*, volume 4149/2006, pages 124–137. Springer Berlin / Heidelberg, 2006.

[38] P. M. E. De Bra and R. D. J. Post. Information retrieval in distributed hypertexts. In *RIAO-94 Conference,*, New York, October 1994.

[39] P. M. E. De Bra and R. D. J. Post. Information retrieval in the World-Wide Web: Making client-based searching feasible. In *First WWW Conference*, Geneva, April 1994.

[40] Inderjit S. Dhillon and Dharmendra S. Modha. Concept decompositions for large sparse text data using clustering. *Machine Learning*, 42(1-2):143–175, 2001.

[41] M. Dorigo and T. Sttzle. *Ant Colony Optimization.* MIT Press, 2004.

[42] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern classification.* New York , Wiley, 2001.

[43] Martin Ester, Hans peter Kriegel, Michael Wimmer, and Xiaowei Xu. Incremental clustering for mining in a data warehousing environment. In *In Proc. 24th Int. Conf. Very Large Data Bases, VLDB*, pages 323–333, 1998.

[44] Oren Etzioni. Moving up the information food chain: Deploying softbots on the world wide web. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence, 1996. Revised version reprinted in AI Magazine special issue, Summer '97. 109*, pages 1322–1326, 1996.

[45] Usama Fayyad, Georges G. Grinstein, and Andreas Wierse, editors. *Information Visualization in Data Mining and Knowledge Discovery.* Morgan Kaufmann, 2001.

[46] Nathaniel Good, J. Ben Schafer, Joseph A. Konstan, Al Borchers, Badrul Sarwar, Jon Herlocker, and John Riedl. Combining collaborative filtering with personal agents for better recommendations. In *In Proceedings of the Sixteenth National Conference on Artificial Intelligence*, pages 439–446. AAAI Press, 1999.

[47] Maria Halkidi, Yannis Batistakis, and Michalis Vazirgiannis. Cluster validity methods: Part i. *ACM SIGMOD Record*, 31(2):40 – 45, June 2002.

[48] Maria Halkidi, Yannis Batistakis, and Michalis Vazirgiannis. Cluster validity checkingmethods: Part ii. *ACM SIGMOD Record*, 31(3):19–27, September 2002.

[49] Jiawei Han and Micheline Kamber. *Data Mining: Concepts and Techniques.* Morgan Kaufmann Publishers, 2001.

[50] J. Handl, J. Knowles, and M. Dorigo. On the performance of ant-based clustering. In *Proceedings of the Third International Conference on Hybrid Intelligent Systems*, 2003.

[51] J. Handl, J. Knowles, and M. Dorigo. Strategies for the increased robustness of ant-based clustering. In *Engineering Self-Organising Systems*, volume Volume 2977/2004 of *Lecture Notes in Computer Science*, pages 90–104. Springer Berlin / Heidelberg, 2004.

[52] Julia Handl and Bernd Meyer. Ant-based and swarm-based clustering. *Swarm Intelligence*, 1(2):95–113, 2007.

[53] S. Havre, E. Hetzler, P. Whitney, and L. Nowell. Themeriver: visualizing thematic changes in large document collections. *Visualization and Computer Graphics, IEEE Transactions on*, 8(1):9–20, Jan/Mar 2002.

[54] S. Haykin. *Neural Networks: A Comprehensive Foundation.* Prentice Hall, 1998.

[55] F. Heppner and U. Grenander. A stochastic nonlinear model for coordinated bird flocks. In S. Krasner, editor, *The Ubiquity of Chaos*, pages 233–238. AAAS, Washington, 1990.

[56] Jonathan L. Herlocker, Joseph A. Konstan, and John Riedl. Explaining collaborative filtering recommendations. In *Proceedings of the 2000 ACM conference on Computer Supported Cooperative Work*, pages 241–250, Philadelphia, Pennsylvania, United States, 2000.

[57] Jonathan L. Herlocker, Joseph A. Konstan, Loren G. Terveen, and John T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst.*, 22:5–53, January 2004.

[58] T. Honkela, S. Kaski, K. Lagus, and T. Kohonen. Newsgroup exploration with websom method and browsing interface. Technical report, Technical Report A32, Helsinki University of Technology, Laboratory of Computer and Information Science, Espoo, Finland,, 1996.

[59] A. K. Jain and R. C. Dubes. *Algorithms for Clustering Data.* Prentice Hall, 1988.

[60] A. K. Jain, M.N. Murthy, and P.J. Flynn. Data clustering: A review. *ACM Computing Reviews*, November 1999.

[61] T. Joachims, D. Freitag, and T. Mitchell. Webwatcher: A tour guide for the world wide web. In *Proceedings of the 1997 IJCAI*, August 1997.

[62] Mehmed Kantardzic. *Data Mining: Concepts, Models, Methods and Algorithms.* Wiley-IEEE Press, 2002.

[63] D.A. Keim. Information visualization and visual data mining. *Visualization and Computer Graphics, IEEE Transactions on*, 8(1):1–8, Jan/Mar 2002.

[64] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings. of the IEEE International Conference on Neural Networks*, volume 4, pages 1942–1948, 1995.

[65] J. Kennedy, RC. Eberhart, and Y. Shi. *Swarm Intelligence.* Morgan Kaufmann Publishers Inc., 2001.

[66] H. K. Kim, J. K. Kim, and Y. H. Cho. A collaborative filtering recommendation methodology for peer-to-peer systems'. In *E-Commerce and Web Technologies.* Berlin / Heidelberg Springer, 2005.

[67] T. Kohonen, S. Kaski, K. Lagus, J. Salojrvi, J. Honkela, V. Paatero, and A. Saarela. Self organization of a massive document collection. *IEEE Transactions on Neural Networks, Special Issue on Neural Networks for Data Mining and Knowledge Discovery*, 11(3):574–585, May 2000.

[68] Joseph A. Konstan, Bradley N. Miller, David Maltz, Jonathan L. Herlocker, Lee R. Gordon, and John Riedl. Grouplens: Applying collaborative filtering to usenet news. *Communications of the ACM*, 40(3):77–87, 1997.

[69] Yehuda Koren. Collaborative filtering with temporal dynamics. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2009.

[70] Raymond Kosala and Hendrik Blockeel. Web mining research: A survey. *ACM SIGKDD Explorations*, 2:1–15, 2000.

[71] I. Kushchu. Web-based evolutionary and adaptive information retrieval. *Evolutionary Computation, IEEE Transactions on*, 9(2):117–125, April 2005.

[72] N. Labroche, N. Monmarche, and G. Venturini. A new clustering algorithm based on the chemical recognition system of ants. In *Proceedings of the 15th European Conference on Artificial Intelligence*, 2002.

[73] N. Labroche, N. Monmarche, and G. Venturini. Antclust: Ant clustering and web usage mining. In *Proc. of GECCO-2003*, pages 25–36. Springer, LNCS, 2003.

[74] N. Labroche, N. Monmarche, and G. Venturini. Web sessions clustering with artificial ants colonies. In *WWW2003, The Twelfth International World Wide Web Conference*, Budapest, Hungary, 2003.

[75] D. D. Lewis and J. Catlett. Heterogeneous uncertainty sampling for supervised learning. In W. W. Cohen and H. Hirsh, editors, *In Proceedings of ICML 94, 11th International Conference on Machine Learning*, pages 148–156. San Francisco, CA: Morgan Kaufmann, 1994.

[76] Henry Lieberman. Letizia: an agent that assists web browsing. In *In 1995 International Joint Conference on Aritifical Intelligence*, pages 924–929, 1995.

[77] D. Lilt and F. Kubala. Online speaker clustering. *Acoustics, Speech, and Signal Processing, 2004. Proceedings. (ICASSP '04). IEEE International Conference on*, 1:I–333–6 vol.1, May 2004.

[78] Bing Liu. *Web Data Mining Exploring Hyperlinks , Contents, and Usage Data.* Springer, 2008.

[79] Bo Liu, Jiuhui Pan, and R I (Bob) McKay. *Incremental Clustering Based on Swarm Intelligence*, pages 189–196. Lecture Notes in Computer Science. in Simulated Evolution and Learning, Springer Berlin / Heidelberg, 2006.

[80] Daben Liu and Francis Kubala. Online speaker clustering. volume 1, pages I–333–6 vol.1, May 2004.

[81] Erik D. Lumer and Baldo Faieta. Diversity and adaptation in populations of clustering ants. In *Proceedings of the third international conference on Simulation of adaptive behavior : from animals to animats 3*, pages 501 – 508. MIT Press Cambridge, MA, USA, 1994.

[82] Kegang Luo, Yuanchao Liu, and Xiaolong Wang. A dynamic som algorithm for clustering large-scale document collection. *Advanced Language Processing and Web Information Technology, 2007. ALPIT 2007. Sixth International Conference on*, pages 15–20, Aug. 2007.

[83] Peter Lyman and Hal R. Varian. How much information 2000. Retrieved from http://www.sims.berkeley.edu/how-much-info on [01.19.2009], 2000.

[84] Peter Lyman and Hal R. Varian. How much information 2003. Retrieved from http://www.sims.berkeley.edu/how-much-info-2003 on [01.19.2009], 2003.

[85] J. MacQueen. Some methods for classification and analysis of multivariate observations. In Lucien M. Le Cam and Jerzy Neyman, editors, *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297. University of California Press, 1967.

[86] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schtze. *Introduction to Information Retrieval.* Cambridge University Press, 2008.

[87] Dieter Merkl and Andreas Rauber. Alternative ways for cluster visualization in self-organizing maps. In *In Proc. of the Workshop on Self-Organizing Maps (WSOM97*, pages 106–111, 1997.

[88] Nancy Miller, Elizabeth G. Hetzler, Grant Nakamura, and Paul Whitney. The need for metrics in visual information analysis. In *Workshop on New Paradigms in Information Visualization and Manipulation*, pages 24–28, 1997.

[89] M. M. Millonas. Swarms, phase transition, and collective intelligence. In *In C. G. Langton, (Eds.), Artificial life III.* Addison - Wesley, MA, 1994.

[90] D. Mladenic. Personal webwatcher: design and implementation. Technical Report Technical Report IJS-DP-7472, School of Computer Science, Carnegie-Mellon University, Pittsburgh, USA, 1996.

[91] B. Mobasher, H. Dai, T. Luo, M. Nakagawa, Y. Sun, and J. Wiltshire. Discovery of aggregate usage profiles for web personalization. In *Proceedings of the WebKDD Workshop*, 2000.

[92] B. Mobasher, H. Dai, T. Luo, Y. Sung, and J. Zhu. Integrating web usage and content mining for more effective personalization. In *In Proc. of the International Conference on E-Commerce and Web Technologies (ECWeb2000)*, Greenwich, UK, September 2000.

[93] Robin R. Murphy. *Introduction to AI Robotics.* The MIT Press, 2000.

[94] O. Nasraoui. World wide web personalization. In J. Wang, editor, *Invited chapter in Encyclopedia of Data Mining and Data Warehousing.* Idea Group, 2005.

[95] O. Nasraoui and C. Petenes C. Combining web usage mining and fuzzy inference for website personalization. In *Proc. of WebKDD 2003 KDD Workshop on Web mining as a Premise to Effective and Intelligent Web Applications*, page 37, Washington DC, August 2003.

[96] O. Nasraoui, C. Cardona, C. Rojas, and F. Gonzalez. Mining evolving user profiles in noisy web clickstream data with a scalable immune system clustering algorithm. In *Proc. of WebKDD 2003*, pages 71–81, Washington DC, Aug 2003.

[97] O. Nasraoui and R. Krishnapuram. One step evolutionary mining of context sensitive associations and web navigation patterns. In *in Proc. SIAM conference on Data Mining*, pages 531–547, Arlington, VA, April 2002.

[98] O. Nasraoui, R. Krishnapuram, H. Frigui, and Joshi A. Extracting web user profiles using relational competitive fuzzy clustering. *International Journal of Artificial Intelligence Tools*, 9(4):509–526, 2000.

[99] O. Nasraoui, R. Krishnapuram, H. Frigui, and Joshi A. Extracting web user profiles using relational competitive fuzzy clustering. *International Journal on Artificial Intelligence Tools*, 9(4):509–526, 2000.

[100] O. Nasraoui, R. Krishnapuram, and A. Joshi. Mining web access logs using a relational clustering algorithm based on a robust estimator. In *8th International World Wide Web Conference*, pages 40–41, Toronto, Canada, 1999.

[101] O. Nasraoui, R. Krishnapuram, and A. Joshi. Mining web access logs using a relational clustering algorithm based on a robust estimator. In *Proc. of the Eighth International World Wide Web Conference*, pages 40–41, Toronto, 1999.

[102] O. Nasraoui, R. Krishnapuram, and A. Joshi. Relational clustering based on a new robust estimator with application to web mining. In *Proceedings of the North American Fuzzy Information Society*, pages 705–709, New York City, 1999.

[103] O. Nasraoui and M. Pavuluri. Complete this puzzle: A connectionist approach to accurate web recommendations based on a committee of predictors. In B. Mobasher, B. Liu, B. Masand, and O. Nasraoui, editors, *In Proceedings of WebKDD- 2004 workshop on Web Mining and Web Usage Analysis, part of the ACM KDD: Knowledge Discovery and Data Mining Conference*, Seattle, WA, 2004.

[104] O. Nasraoui, M. Soliman, E. Saka, A. Badia, and R. Germain. A web usage mining framework for mining evolving user profiles in dynamic web sites. *Knowledge and Data Engineering, IEEE Transactions on*, 20(2):202–215, Feb. 2008.

[105] Olfa Nasraoui. An evolutionary approach to mining robust multiresolution web profiles and context sensitive url associations. *Intl Journal of Computational Intelligence and Applications*, 2:339–348, 2002.

[106] R. Krishnapuram O. Nasraoui. A new evolutionary approach to web usage and context sensitive associations mining. *International Journal of Computational Intelligence and Applications, Special Issue on Internet Intelligent Systems*, 2(3:339–348, Sep. 2002.

[107] S. Goswami O. Nasraoui. Mining and validating localized frequent itemsets with dynamic tolerance. In *Proc. SIAM conference on Data Mining*, Bethesda , MD, April 2006.

[108] M. Omran, A. P. Engelbrecht, and A. Salman. Particle swarm optimization method for image clustering. *International Journal of Pattern Recognition and Artificial Intelligence*, 19(3):297–322, 2005.

[109] M. Omran, A. Salman, and A. P. Engelbrecht. Image classification using particle swarm optimization. In *In Conference on Simulated Evolution and Learning*, volume 1, pages 370–374, 2002.

[110] Luigi Pagliarini, Ariel Dolan, Filippo Menczer, and Henrik Hautop Lund. Alife meets web: Lessons learned. In *VW '98: Proceedings of the First International Conference on Virtual Worlds*, pages 156–167, London, UK, 1998. Springer-Verlag.

[111] M. Pazzani, J. Muramatsu, and D. Billsus. Syskill & webert: Identifying interesting web sites. In *Proceedings of the National Conference on Artificial Intelligence*, pages 54–61, 1996.

[112] Michael Pazzani and Daniel Billsus. Learning and revising user profiles: The identification ofinteresting web sites. *Mach. Learn.*, 27(3):313–331, 1997.

[113] Michael J. Pazzani. A framework for collaborative, content-based and demographic filtering. *Artificial Intelligence Review*, 13(5-6):393–408, 1999.

[114] J. Pearl. Fusion, propagation, and structuring in belief networks. *Artificial Intelligence*, 29(3):241 – 288, 1986.

[115] Mike Perkowitz and Oren Etzioni. Adaptive web sites: an ai challenge. In *In Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, pages 16–23, 1997.

[116] F. Picarougne, H. Azzag, G. Venturini, and C. Guinot. On data clustering with a flock of artificial agents. In *Proceedings of the 16th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'04)*, November 2004.

[117] Fabien Picarougne, Hanene Azzag, Gilles Venturini, and Christiane Guinot. A new approach of data clustering using a flock of agents. *Evolutionary Computation*, 15(3):345–367, 2007.

[118] Frits H. Post, Gregory M. Nielson, and Georges-Pierre Bonneau., editors. *Data Visualization The State of the Art.* Boston ; Dordrecht : Kluwer, 2003.

[119] Beatriz Prieto, Fernando Tricas, Juan J. Merelo, Antonio Mora, and Alberto Prieto. Visualizing the evolution of a web-based social network. *Journal of Network and Computer Applications*, 31(4):677–698, 2008.

[120] Glenn Proctor and Chris Winter. Information flocking: Data visualisation in virtual worlds using emergent behaviours. *Lecture Notes in Computer Science*, 1434:168–176, 1998.

[121] J.R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[122] Vitorino Ramos and Ajith Abraham. Swarms on continuous data. *Evolutionary Computation, 2003. CEC '03. The 2003 Congress on*, 2:1370–1375 Vol.2, Dec. 2003.

[123] Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. Grouplens: An open architecture for collaborative filtering of netnews. In *Proceedings of ACM 1994 Conference on Computer Supported Cooperative Work*, pages 175–186, Chapel Hill, NC, 1994. ACM Press.

[124] Craig W. Reynolds. Flocks, herds, and schools: A distributed behavioral model. *Computer Graphics*, 21(4):25–34, 1987.

[125] Elaine Rich. Users are individuals: individualizing user models. *International Journal of Man-Machine Studies*, 18:199–214, 1983.

[126] C. J. Rijsbergen. *Information Retrieval.* Butterworth-Heinemann, 22nd edition, 1979.

[127] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall Series in Artificial Intelligence, 1995.

[128] Esin Saka and Olfa Nasraoui. Simultaneous clustering and visualization of web usage data using swarm-based intelligence. In *Proceedings of the 20th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'08)*, 2008.

[129] Esin Saka and Olfa Nasraoui. Improvements in flock-based collaborative clustering algorithms. In Christine Mumford and Lakhmi Jain, editors, *Computational Intelligence Collaboration, Fusion and Emergence, Intelligent Systems Reference Library*, pages 639–672. Springer, 2009.

[130] Esin Saka and Olfa Nasraoui. A recommender system based on the collaborative behavior of bird flocks. In *Proceedings of the 6th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom'10)*, 2010.

[131] Badrul M. Sarwar, Joseph A. Konstan, Al Borchers, Jon Herlocker, Brad Miller, and John Riedl. Using filtering agents to improve prediction quality in the grouplens research collaborative filtering system. In *In Proceedings of the 1998 ACM Conference on Computer Supported Cooperative Work*, pages 345–354, Seattle, Washington, 1998.

[132] J. Ben Schafer, Dan Frankowski, Jon Herlocker, and Shilad Sen. Collaborative filtering recommender systems. In *The Adaptive Web, LNCS 4321*, pages 291–324. 2007.

[133] J.B. Schafer, J. Konstan, and J. Reidel. Recommender systems in e-commerce. In *Proceedings of ACM Conference on E-commerce*, pages 158–166, 1999.

[134] A. F. Smeaton and C. J. van Rijsbergen. The retrieval effects of query expansion on a feedback document retrieval system. *The Computer Journal*, 26(3):239–246, 1983.

[135] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction.* A Bradford Book, The MIT Press, Cambridge, Massachusetts London, England, 1998.

[136] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining.* Addison-Wesley, 2005.

[137] Alexey Tsymbal. The problem of concept drift: Definitions and related work. Technical report, 2004.

[138] Edward R. Tufte. *Visual Explanations: Images and Quantities, Evidence and Narrative.* Graphics Press, 1997.

[139] Edward R. Tufte. *The Visual Display of Quantitative Information.* Graphics Press; 2 edition, 2001.

[140] Edward Rolf Tufte. *Envisioning Information.* Graphics Press, 8th printing, June 2001, 1990.

[141] Edward Rolf Tufte. *Beautiful Evidence.* Graphics Press, 2006.

[142] D. W. van der Merwe and A. P. Engelbrecht. Data clustering using particle swarm optimization. In *Proceedings of IEEE Congress on Evolutionary Computation 2003 (CEC 2003)*, volume 1, pages 215– 220, 2003.

[143] Juha Vesanto. Som-based data visualization methods. *Intelligent Data Analysis*, 3:111–126, 1999.

[144] A. L. Vizine, L. N. de Castro, E. R. Hruschka, and R. R. Gudwin. Towards improving clustering ants: an adaptive ant clustering algorithm. informatica, 29, 143154. *Informatica*, 29:143–154, 2005.

[145] Gerhard Weiss, editor. *Multiagent Systems: A Modern Approach To Distributed Artificial Intelligence.* The MIT Press Cambridge, Massachusetts London, England, 2000.

[146] T. White and B. Pagurek. Towards multi-swarm problem solving in networks. In Y. Demazeau, editor, *Proceedings of the 3rd International Conference on Multi-Agent Systems (ICMAS'98)*, Paris, France, 1998. IEEE Press.

[147] Michael Wooldridge and Nicholas R. Jennings. Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 10:115–152, 1995.

[148] T. Yan, M. Jacobsen, H. Garcia-Molina, and U. Dayal. From user access patterns to dynamic hypertext linking. In *Proceedings of the 5th International World Wide Web conference*, Paris, France, 1996.

[149] Y. Y. Yao. Measuring retrieval effectiveness based on user preference of documents. *Journal of the American Society for Information Science (JASIS)*, 46(2):133–145, March 1995.

[150] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. Birch: an efficient data clustering method for very large databases. In *Proceedings of the 1996 ACM SIGMOD international conference on Management of data*, SIGMOD '96, pages 103–114, New York, NY, USA, 1996. ACM.

[151] Zhiyong Zhang. *Focused Image Search in the Social Web.* PhD thesis, University of Louisville, 2008.

CURRICULUM VITAE

NAME: Esin SAKA

ADDRESS: Department of Computer Science and Computer Engineering
University of Louisville
Louisville, KY 40292

EDUCATION:
M.S. Computer Enginering
Middle East Technical University
2005
B.S. Computer Enginering
Middle East Technical University
2002
B.S. Mathematics
Middle East Technical University
2002

PREVIOUS RESEARCH:
Genetic Algorithms
Multi-Agent Systems
Natural Language Processing
Machine Learning

EXPERIENCE:
University of Louisville Research Assistant
Microsoft Research Intern
Yahoo! Inc. Engineering Intern
Middle East Technical University Assistant
The Scientific and Technical Research Council of Turkey

TEACHING:

CECS 121 Program Design in C – TA

CEng561 Artificial Intelligence – TA

CEng334 Operating Systems – TA

CEng140 C Programming – TA

CEng111 Scheme programming – TA

AWARDS:

2011 Graduate Dean's Citation, 2011

Who's Who Among Students, 2010

NSF travel fellowship to attend ICDE 2010

Best graduate poster, KY-WIC, 2010

Google WSDM Conf. Grant and Travel Award, 2009

IEEE Outstanding CECS Student Award, UofL 2008

Yahoo! Sponsored Search Contest 1st Team, 2007

Grace Hopper NSF Scholarship, 2008-2010

Reasoning Web Grant, 2008

Compulog/ALP Summer School on Logic Prog., 2008

CRA-W Grad Cohort, 2006-2008

Google Workshop for Women Engineers, 2006

Turkey Team Deputy Leader, BOI 2002

Milliyet Knowledge Competition Turkey 8th Degree, 1994

PUBLICATIONS:

Book Chapters:

Esin Saka, Olfa Nasraoui, "Improvements in Flock-based Collaborative Clustering Algorithms", book chapter in Computational Intelligence, Collaboration, Fusion and Emergence", Springer Intelligent Systems Reference Library, 2009.

Olfa Nasraoui, Esin Saka, "Web Usage Mining in Noisy and Ambiguous Environments: Exploring the Role of Concept Hierarchies, Compression, and Robust User Profiles", In Berendt, B., Hotho, A., & Semeraro, G. (Eds.).

WebMine 2006, Berlin, Germany, September 2006, Revised Selected and Invited Papers. Springer Lecture Notes in Artificial Intelligence, 2007.

Journal Papers:
Featured article of February 2008 issue: O. Nasraoui, M. Soliman, E. Saka, A. Badia, R. Germain, "A Web Usage Mining Framework for Mining Evolving User Profiles in Dynamic Web Sites", IEEE Transactions on Knowledge and Data Eng., 2008.

Conference Papers:
E. Saka, O. Nasraoui, "A Recommender System Based on the Collaborative Behavior of Bird Flocks", CollaborateCom, 2010.

E. Saka, O. Nasraoui, "On Dynamic Data Clustering and Visualization using Swarm Intelligence", ICDE PhD Workshop, 2010.

Esin Saka, Olfa. Nasraoui, "Dynamic Clustering and Visualization using Swarm Intelligence", GHC09 PhD Forum, 2009.

Esin Saka, Olfa Nasraoui, "Simultaneous Clustering and Visualization of Web Usage Data using Swarm-based Intelligence", Proceedings of the 20th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'08), 2008.

C. Rojas, O. Nasraoui, N. Durak, L. Zuhadar, S. Sellah, Z. Zhang, B. Hawwash, E. Saka, E. Leon, J. Gomez, F. Gonzalez, M. Soliman, "Knowledge Discovery in Data with selected Java Open Source Software, Proceedings of the 2008 Workshop on Building Computational Intelligence and Machine Learning

Virtual Organizations, 2008.

O. Nasraoui , Z. Zhang, and E. Saka , "Web Recommender System Implementations in Multiple Flavors: Fast and (Care) Free for All. In Proceedings of the ACM-SIGIR Open Source Information Retrieval workshop, Seattle , WA , Aug. 2006.

Posters

Esin Saka, "Flock-based Clustering and Data Visualization of Web Clickstreams", presented at Grace Hopper Celebration of Women in Computing (GHC) 2008, Keystone Resort, Colorado, October 1-4, 2008, Kentucky Women in Computing Conference, February 26-27, 2010.

Esin Saka, "The Wind of Flocks for Clustering and Data Visualization ", presented at Engineering Exposition (E-Expo) 2008, Louisville and CRA-W Grad Cohort for Women Program, Seattle, March 13-14, 2008.

Esin Saka, "Intelligent Web Recommender Systems (with a Fast Open Source Implementation) ", presented at CRA-W Grad Cohort for Women Program, San Francisco, March 2-3, 2007.