T. C.
BAHÇEŞEHİR ÜNİVERSİTESİ

NOVEL EFFICIENT AND SCALABLE METHODS FOR COMBINING MULTIPLE CLUSTERINGS

Master of Science Thesis

Arif Murat YAĞCI

Istanbul, 2010

T. C.
BAHÇEŞEHİR ÜNİVERSİTESİ
The Graduate School of Natural and Applied Sciences
Computer Engineering

NOVEL EFFICIENT AND SCALABLE METHODS FOR COMBINING MULTIPLE CLUSTERINGS

Master of Science Thesis

Arif Murat YAĞCI

Supervisor: Asst. Prof. Dr. Selim Necdet MİMAROĞLU

Istanbul, 2010

T. C.
BAHÇEŞEHİR ÜNİVERSİTESİ
The Graduate School of Natural and Applied Sciences
Computer Engineering

Title of the Master’ s Thesis	:	NOVEL EFFICIENT AND SCALABLE METHODS FOR COMBINING MULTIPLE CLUSTERINGS
Name/Last Name of the Student	:	Arif Murat YAĞCI
Date of Thesis Defense	:	13 September 2010

The thesis has been approved by the Graduate School of Natural and Applied Sciences.

Asst. Prof. Dr. Tunç BOZBURA
Acting Director

This is to certify that we have read this thesis and that we find it fully adequate in scope, quality and content, as a thesis for the degree of Master of Science.

Examining Commitee Members:

Asst. Prof. Dr. Selim Necdet MİMAROĞLU :

Assoc. Prof. Dr. Taşkın KOÇAK :

Asst. Prof. Dr. Olcay KURŞUN :

ACKNOWLEDGMENTS

This thesis is dedicated to all the great people who inspired me.

I am so very thankful for my advisor, Dr. Selim Necdet Mimaroglu, who has helped me a lot during my research and also for a future academic research career, which I always wanted. It was a privilege to work with him. His guidance and support has been challenging as well as encouraging which has contributed to my success and well-being.

I thank Dr. Taskin Kocak and Dr. Olcay Kursun for their constructive criticism, time, and support. I also thank Dr. Tevfik Aytekin and other distinguished members of the BSU Engineering Faculty for their help when I needed.

People in my research group have provided great support where I gained more insight into tough problems. I thank other researchers, especially M. Emin Aksehirli and Ertunc Erdil.

ABSTRACT

NOVEL EFFICIENT AND SCALABLE METHODS FOR COMBINING MULTIPLE CLUSTERINGS

YAĞCI, Arif Murat

Computer Engineering

Supervisor: Asst. Prof. Dr. Selim Necdet MİMAROĞLU

September 2010, 93 Pages

Clustering is a semi- or unsupervised process of grouping similar objects together. It is widely used for data understanding and data reduction. Combining Multiple Clusterings is an important research trend in clustering that goes beyond what is typically achieved by a single clustering algorithm. The basic idea is that by taking multiple looks at the same data, one can generate a diverse set of clusterings. By combining these clusterings, it is possible to obtain a *better* Final Clustering or discover some otherwise hidden aspects of the data set. Multiple clusterings may be produced by running different clustering algorithms with varying input parameters. Domain experts, proprietary methods, or a distributed computing environment may provide clusterings. Computationally cheap operations e.g. on random projections or random samplings of a data set may also provide multiple clusterings. A range of applications in Bioinformatics, Computer Vision, and Text Mining, among others, employ algorithms for combining multiple clusterings.

This thesis provides a literature survey and contributes three novel and efficient methods to Combining Multiple Clusterings research. First, we propose a novel binary method for fast computation of an objective function, FastFit, which measures cluster cohesion and separation with respect to object co-associations. This computation method is very efficient in terms of both time and space complexity. Secondly, a novel accurate and scalable consensus method, CLICOM, is proposed to combine multiple clusterings using graph-theoretic cliques. CLICOM employs, as well, a novel output-sensitive clique finding algorithm which works on larger graphs and produces output in a short amount of time. Finally, a set of parallel algorithms is proposed to calculate an approximate distance matrix of a binary data set. These algorithms compute distances by utilizing weak clusterings of randomly hashed objects in shared and distributed memory computing environments.

Experimental results of the proposed methods are shown on synthetic and real data sets. The methods are especially suited to large data sets where efficiency and scalability is a major concern.

Keywords: Clustering, Clustering Ensemble Problem and Consensus Clustering, Binary Methods and Graph Theory, Data Mining, Machine Learning and Pattern Recognition

ÖZET

ÇOKLU BÖLÜMLENMELERİN BİRLEŞTİRİLMESİNDE YENİ VERİMLİ VE ÖLÇEKLENEBİLİR YÖNTEMLER

YAĞCI, Arif Murat

Bilgisayar Mühendisliği

Tez Danışmanı: Yrd. Doç. Dr. Selim Necdet MİMAROĞLU

Eylül 2010, 93 Sayfa

Bölümlenme, benzer veri nesnelerinin yarı denetimli veya denetimsiz şekilde gruplanması işlemidir. Verinin anlaşılması ve indirgenmesinde sıkça kullanılır. Çoklu bölümlenmelerin birleştirilmesi, bölümlenme araştırmalarında önemli bir eğilim olup, tek bir bölümlenme algoritması ile tipik olarak elde edilenden daha ileriye gitmektedir. Temel fikir, aynı veriden farklı bakış açılarıyla değişik bölümlenmelerin yaratılabilmesidir. Bu bölümlenmeler birleştirilerek *daha iyi* bir nihai bölümlenme elde etmek veya verinin daha evvelden saklı kalmış bazı özelliklerini keşfetmek mümkündür. Çoklu bölümlenmeler farklı bölümlenme algoritmalarının değişken giriş parametreleri ile koşturulmasıyla elde edilebilir. Alanının uzmanları, özel mülkiyete tabi yöntemler veya dağıtık bir hesaplama ortamı bölümlenmeler sağlayabilir. Veri kümesinin rastlantısal izdüşümleri veya verinin örneklemeleri üzerinde yapılan az maliyetli hesaplamalar da bölümlenmeler sağlayabilir. Diğerleri yanında özellikle Biyobilişim, Bilgisayarlı Görme ve Metin Madenciliği çoklu bölümlenmelerin birleştirilmesi algoritmalarını kullanmaktadır.

Bu tez bir literatür taraması sağlamakta ve üç yeni ve verimli yöntem ile çoklu bölümlenmelerin birleştirilmesi araştırmalarına katkıda bulunmaktadır. İlk olarak, bölüt iç uyumu ve ayrılığını veri nesnelerinin bölümlenmelerdeki birlikteliklerine göre ölçen bir hedef fonksiyon, FastFit' in hızlı hesaplanması için yeni ikili bir yöntem önerilmektedir. Bu hesaplama yöntemi hem zaman hem de yer karmaşıklığı açısından verimlidir. İkinci olarak, çizge kuramından klikler kullanılarak çoklu bölümlenmelerin birleştirilmesi için yeni hassas ve ölçeklenebilir bir yöntem olan CLICOM önerilmektedir. CLICOM büyük çizgeler üzerinde çalışan ve kısa zamanda sonuç üreten yeni çıktı duyarlı bir klik bulma algoritması da barındırmaktadır. Son olarak, ikili bir veri kümesinin yaklaşık uzaklık matrisini hesaplamak için bir grup paralel algoritma önerilmektedir. Bu algoritmalar, ortak ve dağıtık bellekli hesaplama ortamlarında, rastlantısal olarak çırpı fonksiyonundan geçirilmiş veri nesnelerinin oluşturduğu zayıf bölümlenmeleri kullanarak uzaklıkları hesaplamaktadır.

Önerilen yöntemlerin deneysel sonuçları sentetik ve gerçek veriler üzerinde gösterilmiştir. Yöntemler özellikle verimlilik ve ölçeklenebilirliğin başlıca endişe olduğu büyük verilere uygundur.

Anahtar Kelimeler: Bölümlenme, Bölümlenme Topluluğu Problemi ve Konsensüs Bölümlenme, İkili Yöntemler ve Çizge Kuramı, Veri Madenciliği, Makine Öğrenmesi ve Örüntü Tanıma

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

Approximate Distance Matrix	:	ADM
Adjusted Rand Index	:	ARI
Evidence Accumulation Algorithm	:	EAC
Expectation-Maximization Algorithm	:	EM
Hamming Distance Matrix	:	HM
Locality Sensitive Hashing	:	LSH
Message Passing Interface	:	MPI
Millisecond	:	ms
Minimum Spanning Tree	:	MST
Normalized Mutual Information	:	NMI
Principal Component Analysis	:	PCA
Serial Graph Partitioning Software	:	METIS
Simultaneous Occurence Matrix	:	SOM
Simultaneous Occurence Vector	:	SOV
Sum of Squared Errors	:	SSE

LIST OF SYMBOLS

Clique	:	clq
Clique Matrix	:	$\mathcal{K}$
Cluster (i-th)	:	C_i
Cluster Block	:	B
Clustering of Data Set	:	$\pi(\mathcal{D})$
Consensus Function	:	Γ
Cumulative Pairwise Similarity of a Clique	:	$CECS_{\Pi(\mathcal{D})}(clq)$
Data Set	:	$\mathcal{D}$
Data Set Object (i-th)	:	d_i
Distance between Two Objects	:	$d(\mathbf{u}, \mathbf{v})$
Entropy of i-th Cluster	:	$h(C_i)$
Final Clustering of Data Set	:	$\pi^{\star}(\mathcal{D})$
Inter Cluster Similarity	:	$ECS(\pi(\mathcal{D}))$
Intra Cluster Similarity	:	$ICS(\pi(\mathcal{D}))$
Multiple Clusterings	:	$\Pi(\mathcal{D})$
Mutual Information of Two Clusterings	:	$MI(\pi_1(\mathcal{D}), \pi_2(\mathcal{D}))$
Number of Clusters in Multiple Clusterings	:	$\|\Pi\|_{tc}$
Number of Clusterings in Multiple Clusterings	:	$\|\Pi(\mathcal{D})\|$
Number of Collisions between Two Objects	:	$c(\mathbf{u}, \mathbf{v})$
Objective Function	:	$\phi(\pi(\mathcal{D}))$
Original Class Information of Data Set	:	$\pi^{o}(\mathcal{D})$
Overlapping Percentage	:	δ
Pairwise Cluster Similarity	:	$ECS_{\Pi(\mathcal{D})}(C_k, C_l)$
Random Hashing Function over K	:	f_K
Set of Cliques	:	SC
Set of Near Neighbors of an Object	:	$\mathsf{ANN}(\mathbf{u})$
Similarity Threshold	:	θ
Total Entropy of a Clustering	:	$H(\pi(\mathcal{D}))$

1. INTRODUCTION

This chapter provides information on clustering and combining multiple clusterings research.

1.1 CLUSTERING: BASIC CONCEPTS AND METHODS

Clustering, which is also known as *Unsupervised Classification*, is the process of grouping similar objects together. In other words, clustering is organizing objects in an unpredefined but meaningful manner. The ultimate goal is that the objects within a cluster are similar to each other and different from the objects in other clusters. Clustering is widely used for exploratory data analysis or hypothesizing about data. It may also be used for data reduction by creating an abstraction of data such as data summarization or data compression.

Clustering is an important research area and it is studied extensively in Data Mining, Machine Learning and Pattern Recognition. It is commonly used in disciplines that involve analysis of data, therefore it plays an important role in a wide variety of fields. Natural and Social Sciences ranging from Physics and Biology to Economics as well as many emerging interdisciplinary fields such as Bioinformatics, Information Retrieval, and Computer Vision have important applications of clustering.

The notion of a cluster is not well-defined and the definition of similarity among data objects poses a challenging issue. Therefore, a cluster is a subjective entity that is in the eye of the beholder and its significance and interpretation requires domain knowledge Jain (2010). Figure 1.1 shows that clusters can differ e.g. in shape, size, and density. Many factors, such as noise significantly affect the accuracy of a clustering algorithm.

Although humans are excellent cluster seekers in two and possibly three dimensions, computer algorithms are needed to cluster sophisticated and high-dimensional data sets where curse of dimensionality Bellman (2003) is added to the existing set of issues in clustering.

Yet another problem in clustering is the unknown number of true natural clusters for the given data. Most clustering algorithms take number of clusters as an input parameter.

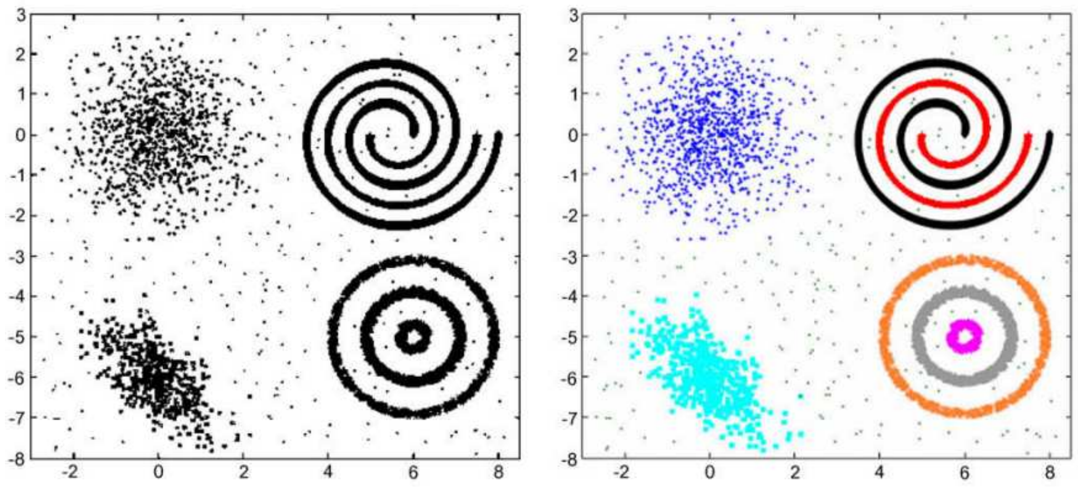


Figure 1.1: Left Figure shows input data. Right Figure shows a desired clustering with 7 different clusters and isolated noise. Clusters differ in shape, size, and density. (Figure taken from Jain (2010))

More detailed information on clustering can be found in Tan et al. (2005), Han and Kamber (2005), Alpaydin (2010), Jain and Dubes (1988), Jain (2010).

1.1.1 Clustering Methods

There is no single clustering algorithm that works well on all the data sets. Kleinberg (2002) suggests a formal perspective based on three criteria to show impossibility of finding a clustering function which satisfies this formalism. Finding natural groupings of a data set is a hard to accomplish task: Thousands of clustering algorithms in the literature speak to this nature. This is also due to the fact that there is no *best* clustering algorithm. Each clustering technique makes some assumptions about the underlying data set. If the assumptions hold, good clusterings can be expected. But, more than often assumptions about the data set do not hold, which in turn means bad clusterings will be generated.

It is difficult to provide a crisp categorization of clustering methods, because quite often these categories overlap, so that a method may have features from several categories. Han and Kamber (2005) provide a useful categorization of clustering methods. In the following, basic clustering concepts and some influential clustering algorithms are presented.

Partitional vs. Hierarchical Clustering : Partitional clustering methods divide data into non-overlapping clusters such that each data object is in one cluster. On the other hand, Hierarchical clustering methods permit clusters to have subclusters and therefore form a tree of nested clusters. Leaf nodes of the tree represent data set objects as singleton clusters, and the root represents a cluster containing all the objects. Each intermediate node (cluster) is the union of its children. A partional clustering can be obtained by cutting the tree at a particular level.

Exclusive vs. Fuzzy Clustering : A clustering is exclusive or hard if each object is assigned to a single cluster. An overlapping or non-exclusive clustering refers to one, in which an object can belong to more than one cluster. In Fuzzy Clustering, clusters are treated as Fuzzy Sets (see Lee (2004)), where each object belongs to every cluster with a membership weight between 0 and 1.

Complete vs. Partial Clustering : In a complete clustering, every object is dutifully assigned to some cluster. A partial clustering algorithm does not necessarily do this assignment since some objects may not belong to well-defined groups and represent e.g. noise or outliers.

Different notions of a Cluster : A cluster is *well-separated* if each data object is more similar to every other object in the cluster than to any object not in the cluster. If a clustering consists of well-separated clusters, then the distance between any two points in different clusters is larger than the distance between any two points within a cluster. At another direction, it is useful to make a *prototype-based* definition of a cluster where a prototype refers to some statistical representation such as a centroid or medoid. Representing data as a graph allows *graph-based* notion of a cluster, where a cluster can be seen as a connected component based on connectivity of graph vertices. A problem with this approach may occur when noise is present. Another strong graph-based approach defines a cluster as a clique. Chapter 3 uses this approach for combining multiple clusterings. *Density-based* approach defines a cluster as a dense region of objects surrounded by a region of low density. This approach is often employed when the clusters are intertwined or not regular, and noise and outliers are present. Yet another definition of a cluster is a *conceptual cluster* which encompasses above mentioned definitions. The specific concept of a cluster is needed to be supplied to the clustering algorithm. It is a challenging problem to create the notion of a cluster, as the concept gets too sophisticated.

k-Means

k-Means is a seminal clustering algorithm which finds k prototype-based clusters (represented by centroids), where k is a user-defined input. Each data object is assigned to the most similar cluster. The basic algorithm is partitional and the final clustering is exclusive and complete. There are many variations of k-Means in the literature including its fuzzy version, Fuzzy C-Means. Jain (2010) provides a detailed survey on k-Means clustering. A similar algorithm, k-Medoids, works with cluster medoids instead of centroids.

```
Input: $\mathcal{D}$: Data Set, $k$: Number of Clusters
Output: $k$ Clusters
Randomly select $k$ points as initial centroids ;
repeat
    Form $k$ clusters by assigning each point to the closest centroid ;
    Recompute $k$ new centroids ;
until Centroids do not change ;
return $k$ Clusters;
```

Algorithm 1: Basic k-Means Algorithm

k-Means clustering can also be stated as an optimization problem with an objective to minimize a sum of squared error function, $SSE = \sum_{i=1}^{k} \sum_{d \in C_i} distance(c_i, d)^2$ where distance function is based on Euclidean distance, d is a data set object, c_i is a centroid, and C_i is the cluster that centroid represents.

Hierarchical Clustering

Hierarchical Clustering algorithms can be agglomerative or divisive. They generate a hierarchy of nested clusters that can be represented by a binary tree, called a *dendrogram*. Agglomerative hierarchical clustering algorithms start with each data point as an individual cluster and merge the most similar pair of clusters iteratively. The dendrogram is cut at a certain level to obtain meaningful clusters. The basic algorithm creates an exclusive and complete clustering.

Definition of cluster similarity differs in various agglomerative hierarchical clustering techniques. *Single Link* defines cluster similarity as the similarity between closest two points that are in different clusters whereas *Complete Link* defines it as the similarity of farthest two points in different clusters. *Group Average* defines cluster similarity to be the average pairwise similarities of all pairs of points from different clusters. All these

```
Input: $\mathcal{D}$: Data Set
Output: Dendrogram
Compute similarity matrix ;
Initialize Dendrogram with each data set object being a singleton cluster ;
repeat
    Merge two most similar clusters ;
    Update similarity matrix with the new cluster ;
    Update Dendrogram ;
until All clusters are merged ;
return Dendrogram;
```

Algorithm 2: Basic Agglomerative Hierarchical Clustering Algorithm

approaches are graph-theoretic. An alternative technique, *Ward' s method* assumes that a cluster is represented by its centroid and measures similarity between two clusters in terms of decrease in the SSE that results from merging the clusters.

Besides their usefulness, Agglomerative Hierarchical Clustering algorithms are expensive in terms of their computational and storage requirements especially with large data sets. Once a cluster is formed, there is no going back. It may be problematic to decide where to cut the dendrogram. It is hard to tell the inner structure of a cluster from the dendrogram, e.g. which object is the medoid of the cluster and which objects are the borders of the cluster. These algorithms are sensitive to small perturbations in the data.

Other Important Clustering Algorithms

DBSCAN Ester et al. (1996) is an influential density-based clustering algorithm. It classifies a data set object as one of the cores of a cluster, if the object has more than $minPts$ neighbors within an ϵ neighborhood. Clusters are formed by connecting neighboring core objects and non-core objects either serve as the boundaries of clusters or classified as noise. Since noise is typically randomly distributed, density of a cluster should be significantly higher than that of noise. DBSCAN finds arbitrarily-shaped clusters and unlike k-Means or hierarchical methods, it forms partial clusterings since it may mark some data set points as noise. DBSCAN can be problematic with clusters of varying densities. Finding the proper $minPts$ and ϵ is another issue. High dimensional data causes problems because it is difficult to recognize densities in such dimensions. A grid-based clustering technique CLIQUE Agrawal et al. (1998) and a kernel-based scheme DENCLUE Hinneburg and Gabriel (2007) are also examples of density-based clustering.

Graph-based clustering techniques construct a weighted graph from data, where weights represent similarities between data set objects or some higher level view of data. Minimum Spanning Tree (MST) clustering finds MST of the dissimilarity graph and cuts edges with largest dissimilarity iteratively until singleton clusters remain. This is equivalent to Hierarchical clustering algorithm with Single Link. OPOSSUM Strehl et al. (2000) and CHAMELEON Karypis, Han and Kumar (1999) are some of the important graph-based techniques. METIS Karypis and Kumar (1998a) is not a clustering algorithm itself, but it is often used in graph-based clustering to partition the graph in an efficient way. In Chapter 3, we give more detail about graph-based clustering and propose a graph-based method for combining multiple clusterings.

Prototype-based notion of a cluster is used in k-Means in a simple but effective way. This notion is commonly used in Machine Learning and Pattern Recognition. Mixture Model Clustering is more general than k-Means, because it deals with various types of statistical distributions. Many real life data sets are indeed the result of random processes, and thus should satisfy the statistical assumptions of these models. Mixture models view data as a set of observations from a mixture of different probability distributions. Finding parameters of each distribution means location of a cluster. Expectation-Maximization (EM) algorithm Dempster et al. (1977) is commonly used to estimate mixture model parameters using maximum likelihood principle. In the end, a data set object is assigned to each distribution with some probability. Self-Organizing Maps (SOM) Kohonen et al. (2001) can also be considered a prototype-based clustering technique from a neural network viewpoint. SOM assigns each data set object to the cluster whose centroid provides the best approximation of it.

Semi-supervised clustering algorithms Chapelle et al. (2006) utilize any *side information* available along with the data set and the similarity matrix. For example, a must-link constraint specifies that a pair of data set objects connected by the constraint belong to the same cluster whereas a cannot-link constraint specifies the opposite. Such constraints can be provided by the domain expert or domain ontology.

1.1.2 Cluster Validation

Almost every clustering algorithm will dutifully find clusters, even if the data set has no natural clusters. Figure 1.2 shows clustering results of k-Means, DBSCAN, and Single Link Hierarchical Clustering algorithms respectively on a data set with uniformly distributed data set objects.

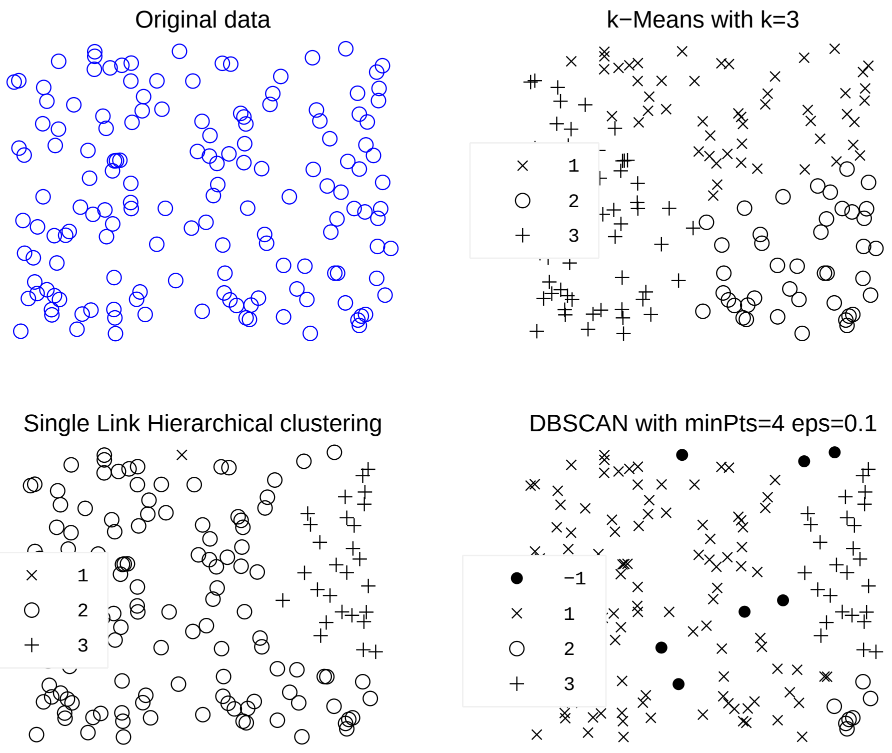


Figure 1.2: Clustering 150 uniformly distributed data set objects

So, what is a good clustering? Cluster Validation refers to a formal evaluation of how good a clustering is. It is a challenging topic and some even argue that Cluster Validation is unnecessary since clustering is of exploratory nature and the notion of a cluster differs from one algorithm to another. Nonetheless, there are many cluster validation measures in the literature. Jain and Dubes (1988), Tan et al. (2005) classify validation measures, or indices into three types:

Supervised (External) Methods : validate a cluster against some a priori external information. Often, this information is the true class memberships. *Entropy* is an information-theoretic approach for measuring cluster validity. Given a data set $\mathcal{D}$ and a clustering, $\pi(\mathcal{D}) = \{C_1, C_2, \ldots, C_{|\pi|}\}$, entropy of each cluster, C_i, is calculated using the formula $h(C_i) = -\Sigma_{j=1}^{|\pi^o(\mathcal{D})|} \frac{|C_j^o \cap C_i|}{|C_i|} \log_2 \frac{|C_j^o \cap C_i|}{|C_i|}$, where $\pi^o(\mathcal{D}) = \{C_1^o, C_2^o, \ldots, C_{|\pi^o|}^o\}$ represents the true class memberships (ground truth). Total entropy of $\pi(\mathcal{D})$ is used to evaluate its validity and calculated using the formula $H(\pi(\mathcal{D})) = \Sigma_{i=1}^{|\pi(\mathcal{D})|} \frac{|C_i|}{|\mathcal{D}|} h(C_i)$. *Purity*, *Precision*, *Recall*, and *F-Measure* are some of the other well-known supervised validation methods that evaluate the extent to which a cluster contains objects of a single class. A group of supervised validation methods, on the other hand, measure to extent which two objects that coexist in the same class also coexist in the same cluster. It is possible to construct two $\mathcal{D} \times \mathcal{D}$ binary matrices for cluster and class similarities respectively where 1's represent coexistence of two data set objects. *Correlation* between these two matrices is used as a validity measure. *Rand Index* Hubert and Arabie (1985) and well-known *Jaccard Index* evaluate cluster validity by looking at a contingency table for number of coexistences in classes/clusters. In Chapter 3, we use *Adjusted Rand Index* and an entropy-based method for cluster validation.

Unsupervised (Internal) Methods : measure the goodness of a clustering with respect to the data set itself or the similarity matrix. Some of these measures are based on *Cluster Cohesion* which determines how similar the objects are in a single cluster. Some are based on *Cluster Separation* which measures how isolated or well-separated a cluster is from the others. There are graph-based and prototype-based approaches to Cohesion and Separation Tan et al. (2005). Chapter 2 presents a graph-based hybrid approach combining cohesion and separation for validation of combined multiple clusterings. Another hybrid method is Silhoutte coefficients Kaufman and Rousseeuw (2005) which calculates a coefficient for each data set object, based on its similarity with all other objects in its cluster and all other clusters. Apart from cohesion and separation, the similarity matrix can directly be used to validate a clustering by reordering its columns and rows according to the class labels in the clustering. This matrix must ideally be a block diagonal structure. Correlation between ideal and actual matrices can be evaluated. The method also enables visual inspection of clusters, but it may be costly for large data sets.

Relative Methods : are not specific methods for cluster validation, but rather refer to a methodology to decide which validity methods are better for the problem in some sense. Any cluster validity measure can be used as a relative measure. Significance of different measures for available clusterings to be validated needs to be assessed. Vendramin et al. (2009) provide a comparison of relative clustering validity criteria.

1.2 COMBINING MULTIPLE CLUSTERINGS

Combining Multiple Clusterings, which is also known as *Consensus Clustering* or *Clustering Ensemble problem*, refers to combining available information in existing clusterings (partitions) of a data set into a new final clustering. The basic idea is that by taking multiple looks at the same data, one can generate diverse clusterings. By combining these clusterings, it is possible to obtain a *better* Final Clustering or discover some otherwise hidden aspects of the data set.

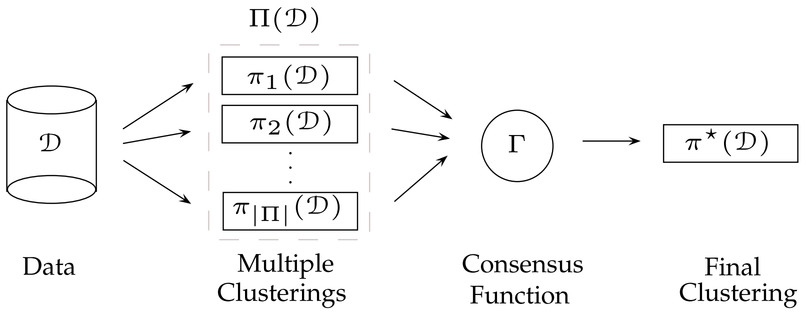


Figure 1.3: An abstract view of Combining Multiple Clusterings

The success of ensemble methods in sensor fusion and classification (supervised learning) problems Kittler et al. (1998), Dietterich (2000), Lam (2000), has motivated the development of ensemble methods for clustering problems (see Fred and Jain (2005)). Influential classifier ensemble algorithms like Bagging Breiman (1996) and Boosting Freund and Schapire (1997) can combine classification results from different classifiers successfully to obtain more accurate results. However, Clustering Ensemble problem is a harder one, mainly due to its nature and the *class correspondence* problem.

Another motivation comes from an application point of view. Diversity of clustering techniques motivates creating multiple clusterings of a data set and combine their results for a better clustering. On the other hand, clustering algorithms take some input parameters. It may be of interest to run a clustering algorithm with varying input parameters to obtain multiple clusterings. Creating multiple clusterings by random samplings or random projections of a data set is used widely. Distributed computing enables creation of multiple clusterings at different sites by using task- or data-parallelism. A combination of clustering results are necessary. Clusterings provided by domain experts, or semi-supervised clustering algorithms may be introduced in an ensemble as an additional clustering. Finally, either the data set or the clustering methods can be proprietary, and only some clustering results are made available. Section 1.2.5 provides real life applications.

Topchy, Jain and Punch (2004) suggest that Combining Multiple Clusterings can go beyond a single clustering algorithm in several respects :

- *Robustness :* Better average performance across the domains and data sets
- *Novelty :* Finding a new combined solution which is better or reveals some otherwise hidden aspects of data
- *Stability and Confidence :* Less sensitivity to noise, and outliers. Clustering uncertainty due to cluster overlappings may be resolved.
- *Parallelization and Scalability :* Ability to integrate solutions from multiple distributed sources.

Figure 1.3 illustrates an abstract view of Combining Multiple Clusterings. A set of clusterings, $\Pi(\mathcal{D})$, is generated from a data set, $\mathcal{D}$. These clusterings are usually represented by the cluster memberships of data set objects. In some cases, they may be represented by their cluster prototypes, or such information may be available as side information as well. The clusterings are then combined into a new final clustering, $\pi^{\star}(\mathcal{D})$, using a *consensus function*, Γ. Chapters 2 - 4 provide a more detailed view of combining multiple clusterings for the specific methods they present.

A variety of consensus functions are defined in the literature. In an influential paper by Fred and Jain (2005), the consensus function, Γ, is based on a *co-association measure* defined by $coassoc(d_i, d_j) = votes_{ij}/|\Pi(\mathcal{D})|$, where $votes_{ij}$ is the number of co-occurrences of data objects d_i and d_j in a cluster, such that $d_i \in \mathcal{D}$ and $d_j \in \mathcal{D}$. The co-associations are held in a $\mathcal{D} \times \mathcal{D}$ matrix which is used as an input to hierarchical and graph-based clustering algorithms. This approach operates at object resolution. Strehl and Ghosh (2002) propose consensus functions, where the set of multiple clusterings, $\Pi(\mathcal{D})$, is represented by a *hypergraph*. These solutions typically operate at lower resolution, namely cluster level, where hyperedges represent clusters. Filkov and Skiena (2004), Cristofor and Simovici (2002) propose consensus functions based on *median partition* approach. The consensus clustering process is defined as finding a new clustering (median partition), $\pi^{\star}(\mathcal{D})$, which minimizes the function $\Psi = \Sigma_{i=1}^{\Pi(\mathcal{D})} diff(\pi_i(\mathcal{D}), \pi^{\star}(\mathcal{D}))$ where $diff$ is a function measuring difference between two clusterings. The median partition approach is shown to be NP-complete in Barthelemy and Leclerc (1995), therefore, it is commonly used in optimization-based approaches to Combining Multiple Clusterings. $diff$ is defined e.g. info-theoretically as in Strehl and Ghosh (2002), Cristofor and Simovici (2002) or based on co-associations of objects as in Filkov and Skiena (2004).

1.2.1 Methods for Combining Multiple Clusterings

The problem of Combining Multiple Clusterings, is an active research topic in Data Mining, Machine Learning, and Pattern Recognition. In this section, we provide an overview of some notable methods in the literature. Table 1.1 summarizes these methods. Chapter 3 provides more details about some of the methods used for comparison with the proposed method in that chapter. Application-specific solutions are presented in Section 1.2.5.

There are graph-based, prototype-based, and optimization-based approaches for combining multiple clusterings. Some of the methods work on objects, whereas others work with higher level representations of data such as clusters. Unfortunately most of the proposed methods do not scale well on large data sets.

Fred and Jain (2005) propose Evidence Accumulation (EAC) for combining multiple clusterings. This method builds a co-association matrix of data set objects and then applies hierarchical clustering to this matrix using either Single Link or Group Average as measures of cluster similarity. There are also other graph-based methods that directly operate on the co-association matrix. Li et al. (2007) propose a hierarchical clustering technique that uses a heuristic called Normalized Edges to merge two similar clusters. Strehl and Ghosh (2002) propose CSPA algorithm which takes this matrix as input and divides data into k partitions by applying METIS partitioning algorithm to it. Luo et al. (2006) apply spectral clustering techniques to this matrix.

Some graph-based techniques work on clusters that constitute the Multiple Clusterings, rather than data set objects. Strehl and Ghosh (2002) propose two algorithms that work in this manner. HGPA views Multiple Clusterings as a hypergraph, and clusters as hyperedges. Note that a hypergraph is a generalization of a graph, where an (hyper)edge connects a subset of its vertices. HGPA runs HMETIS Karypis, Aggarwal, Kumar and Shekhar (1999) on the hypergraph to partition it into k clusters. The other algorithm, MCLA, takes the hypergraph perspective as well and merges hyperedges in k so-called metaclusters. Details of both algorithms are provided in Chapter 3. Hore et al. (2009) propose Bipartite Merger (BM), and METIS Merger (MM), two graph-based techniques that work on cluster prototypes.

Topchy, Jain and Punch (2004) propose a prototype-based mixture model for combining multiple clusterings. Each data set object is represented by a vector of categorical variables corresponding to class labels generated by a set of input clusterings, $\Pi(\mathcal{D})$.

These categorical variables are viewed as the outcome of a multinomial trial. A final clustering is modeled with k components in a mixture of multinomial distributions and the Expectation-Maximization algorithm (EM) is used to optimize latent model parameters for each component to decide final cluster labels. Hongjun et al. (2009) propose a Bayesian ensemble model and employ EM for combining multiple clusterings.

Many solutions treat combining multiple clusterings problem as an optimization problem. These solutions try to minimize or maximize some objective function to find the final clustering. Strehl and Ghosh (2002) see the problem as a combinatorial optimization problem and propose an information-theoretic objective function, Normalized Mutual Information, NMI, to find the median partition (final clustering). They also devise a greedy algorithm which tries to minimize this objective function. Chapter 3 provides details of this function, since it is commonly used as a cluster validity measure in the literature. Another information-theoretic objective function, Quadratic Mutual Information, QMI, is proposed by Topchy et al. (2005). Several methods exploit Evolutionary Algorithms. Cristofor and Simovici (2002) use Genetic Algorithm (see Michalewicz (1996)) to find the median partition. An entropy-based function is defined to evaluate similarity of two clusterings. The objective is to minimize sum of dissimilarities between the median partition and all clusterings in the set of multiple clusterings. This corresponds to the fitness of the median partition. Mohammadi et al. (2008) also use Genetic Algorithm. Fitness of a clustering in the population is evaluated by weighted sums of total intra cluster similarities and pairwise inter cluster similarities which are based on object co-associations. Wolfgang et al. (2000) propose a Hamming-distance based fitness function to evaluate binary-encoded chromosomes (clusterings). Filkov and Skiena (2004) try to find the median partition using Simulated Annealing (see Kirkpatrick et al. (1983)) and use a Rand Index based heuristic to calculate similarities between clusterings. Tumer and Agogino (2008) also use a Simulated Annealing based approach and propose an adaptive version of NMI. Azimi et al. (2009), Yang et al. (2006) propose methods based on Swarm Intelligence (see Abraham et al. (2006)) such as Ant Colony Optimization.

1.2.2 Factors Affecting Final Clustering Quality

All above-mentioned methods will eventually combine a set of multiple clusterings into a new final clustering. However, is it for sure that a better or useful final clustering is obtained all the time? The answer depends on the following factors affecting the Final

Table 1.1: Some notable methods for Combining Multiple Clusterings

Method	**Approach**	**Advantages**	**Shortcomings**
EAC	Hierarchical clustering performed on co-association matrix	All advantages of Hierarchical clustering on the powerful co-association matrix e.g. Single Link finds contiguity-based clusters	Performs poorly with large datasets, an issue arises as to where to cut the dendrogram, no going back once data set object is clustered
CSPA	Graph partitioning performed on co-association matrix	METIS works much faster than Hierarchical clustering, it also finds good quality clusters	Performs poorly with large datasets, user has to specify number of clusters
HGPA	Runs HMETIS on a hypergraph representation of Multiple Clusterings	Very fast and scalable	Poor accuracy
MCLA	Merges similar clusters in meta-clusters	Fast and scalable (uses METIS on cluster based representation of data)	Cluster similarity based on Jaccard similarity, user has to specify number of clusters
EM-Topchy	Models Final Clustering as a mixture and uses EM to estimate model parameters	Detects non-spherical cluster models, co-association matrix not needed	Computational requirements may be high, Fixed number of clusters required in input clusterings and the Final Clustering
Greedy NMI	Combinatorial Optimization to find median partition	No co-association matrix, proposes a widely used heuristic	Very inefficient in terms of computation time
GACEII	Genetic Algorithm using weighted intra- and inter-cluster similarities	Good or compatible accuracy with most approaches, Automatic detection of number of clusters	Storage of co-association matrix, costly operations to calculate fitness
ACO-Azimi	Ant Colony Optimization	Automatic detection of number of clusters, outlier detection	Storage of co-association matrix, run-time results with large datasets not shown

Clustering quality;

Quality of Clusterings : A data set can be clustered in many ways depending on the clustering algorithm employed or how algorithm parameters are set. Such clusterings constitute a set of multiple clusterings, which is usually the only information available to obtain a new final clustering. If the clusterings are not good quality i.e. the information they contain are very diverse and noisy, it may be hard to improve the final clustering quality. Problems reveal especially with high-dimensional large datasets where obtaining good single clusterings may be challenging.

Number of Clusterings : Recent studies have empirically demonstrated improved accuracy of clustering ensembles on a number of artificial and real-world data sets. Unlike certain multiple supervised classifier systems, convergence properties of consensus functions are not well studied. Topchy, Law, Jain and Fred (2004) present formal arguments on the effectiveness of combining multiple clusterings from several perspectives. Consensus functions based on stochastic partition generation, re-labeling, voting and, median partition approaches are shown to converge to a true underlying clustering solution as the number of partitions in the ensemble increases. On the other hand, too many clusterings may cause overfitting.

Choice of Consensus Function : This may be important for specific problems. Some exemplary situations are as follows. Optimization-based solution may not converge with large datasets. Graph-based methods where clusters are represented by vertices may be less efficient, if cluster sizes vary a lot.

1.2.3 Final Clustering Validation

Validation of the final clustering, $\pi^{\star}(\mathcal{D})$, combined from Multiple Clusterings, $\Pi(\mathcal{D})$, can be important in many applications. Moreover, cluster validation may even be at the heart the clustering algorithm (see Strehl and Ghosh (2002), Mohammadi et al. (2008)).

Average NMI (ANMI) is an unsupervised validity measure presented in Strehl and Ghosh (2002) and used as part of their greedy algorithm explained in Section 1.2.1. Details of NMI can be found in Chapter 3. ANMI is defined as $\frac{1}{|\Pi(\mathcal{D})|}\Sigma_{i=1}^{|\Pi(\mathcal{D})|}NMI(\pi^{\star}(\mathcal{D}), \pi_i(\mathcal{D}))$, which takes the average of all NMIs between the Final Clustering and each $\pi_i(\mathcal{D}) \in \Pi(\mathcal{D})$. Fred and Jain (2005) improve this measure by making it sensitive to number of clusters in the Final Clustering, relaxing the assumption that the number of clusters in the Final Clustering are already known.

Chapter 2 defines FastFit which can be used as an unsupervised validity measure with appropriate parameterization. This measure evaluates cluster separation (inter cluster similarity) and cluster cohesion (intra cluster similarity) based on co-association of data set objects. It should be noted that, separation and cohesion can also be used as separate measures of validity.

Fred and Jain (2008) propose another unsupervised validity measure. Initially, a probabilistic clustering model is derived from Multiple Clusterings. The validity corresponds to a minimum description length for both estimated model parameters and the Final Clustering.

A recent paper by Duarteo et al. (2010) propose modified versions of three well-known clustering validity measures for evaluating final clustering validity.

1.2.4 Combining Weak Clusterings

Topchy et al. (2003) provide the following definition; a *weak clustering algorithm* produces a clustering, which is only slightly better than a random one. Such algorithms are usually very simple and computationally inexpensive. Nonetheless, combining their output clusterings can be more attractive than combining clusterings obtained by more sophisticated, but computationally demanding algorithms.

Topchy et al. (2005) argue that combining multiple weak clusterings still achieve comparable or in some cases better performance. The motivation is that the synergy of many such clusterings will compensate for their weaknesses. They propose two methods for obtaining such clusterings. The first method is based on random projection of a data set to a subspace of its attributes. An example would be projecting data on 1-D and applying k-Means on this projection. Second method splits data randomly by hyperplanes. For example, a single random hyperplane creates a trivial clustering by cutting the hypervolume into two. In addition to some theoretical basis, they also provide experimental results with several data sets using different numbers of input clusterings and projection sizes.

Fern and Brodley (2003) approach the problem from a high dimensional clustering point of view. Distance-based clustering of high dimensional data poses a challenging problem, since in a high dimensional space, data tend to be sparse Bellman (2003). They create multiple clusterings of high dimensional data sets by first random projection and then also by Principal Component Analysis (PCA). They show empirically, by using different consensus functions, that multiple clusterings generated by random projections yield good final clusterings.

Minaei-bidgoli et al. (2004) create multiple clusterings by using subsamples of a given data set. In their empirical study, they sample a data set with replacement (bootstrap) and without replacement, and then create clusterings on these views of the data set. A meaningful final clustering for the entire set of data points emerges from multiple clusterings of subsamples. Depending on the consensus function, clusterings of subsamples either update a co-association matrix or provide prototype-based clusters which are further processed to obtain a final clustering.

In Chapter 4, a novel parallel algorithm is introduced, which creates multiple clusterings based on random projections of a binary data set and the resultant co-association matrix correlates with Hamming distance matrix.

1.2.5 Applications

This section presents some applications and application-specific approaches to Combining Multiple Clusterings in the literature that are common in Bioinformatics, Computer Vision, and Text/Document Clustering. A few applications are reported in Medicine, Finance, and Weather Prediction. Applications for specific data formats are also available.

Bioinformatics : Human genome project has created a huge amount of data. Clustering is extensively used in mining gene expression data sets to find important genetic and biological information. Obtaining high quality clusterings is often very challenging due to inconsistent results of different clustering algorithms and noisy data. It is a daunting task for researchers to choose, if available, the best clustering algorithm and generate the best clustering results for their data sets. This is exactly where Combining Multiple Clusterings come in handy. Hu and Yoo (2004), Hu et al. (2006) provide a clustering ensemble framework, GE-Miner, by adapting several graph-based and prototype-based methods for gene expression data. Yu et al. (2007) propose a graph-based consensus function for microarray data, in which data is subsampled a number of times to create multiple clusterings. A co-association matrix is constructed from these input clusterings which is then partitioned using the normalized cut algorithm Shi and Malik (2000) into a final clustering. This process is repeated iteratively and each time the resulting final clustering quality is evaluated by a modified version of Rand Index. Iam-on et al. (2010) propose an algorithm for larger gene data sets, in which a graph-based method work on a clusterwise similarity matrix. Asur et al. (2007) propose a clustering ensemble framework for clustering protein–protein interaction networks, where they apply either a recursive-bisection or hierarchical clustering algorithm on a cluster membership matrix representing input clusterings, whose dimensions are reduced by PCA. Some of the other notable methods are presented in Avogadri and Valentini (2009), Faceli et al. (2009).

Computer Vision : Image segmentation can be considered as a process of clustering pixels in a digital image to obtain meaningful pixel groupings or to locate regions or objects. Segmentation of nontrivial images is one of the most difficult tasks in image processing Gonzalez and Woods (2006). A method for discovering brain lesions in medical images is proposed by Li et al. (2006) in which they combine input clusterings generated by a modified DBSCAN algorithm and show that it is superior to diagnosis results of a single clustering algorithm. Chang-ming et al. (2008) segment ultrasound images by creating a co-association matrix of the various input clusterings and then applying spectral clustering on it. They compare their results with other clustering methods including the domain expert's. In another interesting paper, Gllavata et al. (2006) detect static text in videos using a fuzzy clustering ensemble technique. This technique does not work on a single image, but rather takes incremental snapshots and include temporal information in multiple clusterings. Figure 1.4 illustrates the overall algorithm. Each frame is divided into blocks, these blocks are clustered by Fuzzy C-means and the clustering results generate a set of multiple clusterings. Silva and Scharcanski (2010) use Combining Multiple Clusterings to improve segmentation results in motion tracking. A common problem in satellite imagery is multisource image analysis, where different sensors provide images of the same location. Forestier et al. (2008) propose a method called collaborative clustering which combine results of different clusterings into a final clustering which represents the improved segmentation result. Some of the other interesting applications in computer vision are given in Kyrgyzov et al. (2007), Zhang, Jiao, Liu, Bo and Gong (2008), Ma et al. (2009), Chang et al. (2008), Elhadary et al. (2007).

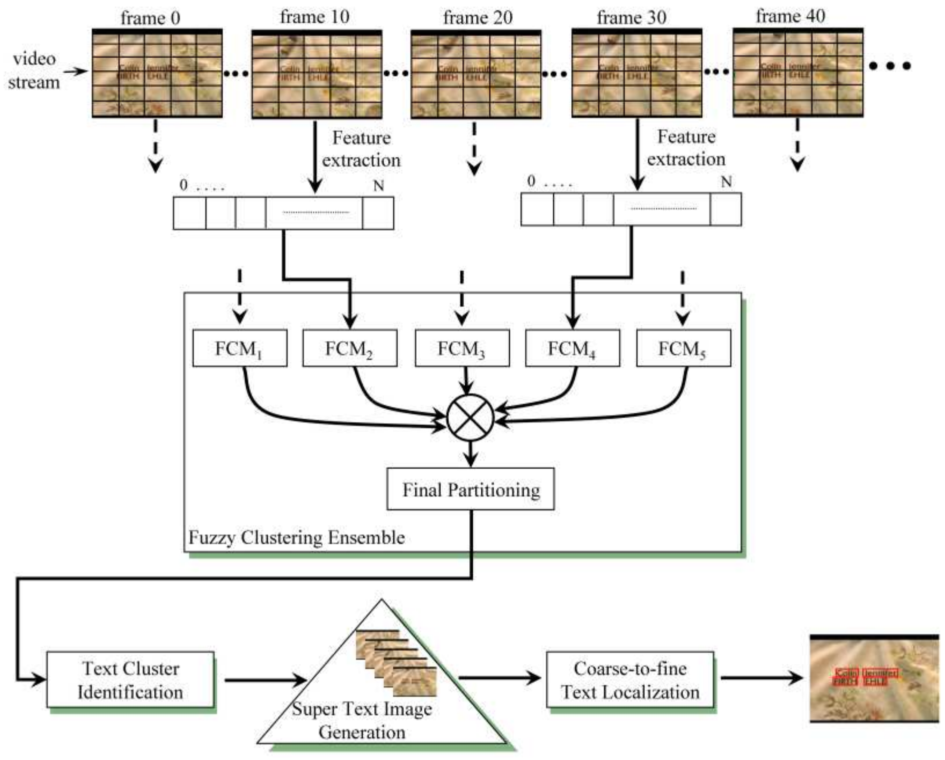


Figure 1.4: Text in video is successfully segmented by combining multiple clusterings. (Figure taken from Gllavata et al. (2006))

Text/Document Clustering : Clustering texts and documents is one of the important tasks in Text Mining. This kind of clustering is quite challenging since text and document databases are usually large, high-dimensional, and natural clusters in such databases overlap. Gonzalez and Turmo (2008) present an empirical comparison of the effectiveness of two different strategies for the generation of clustering ensembles. The first one relies on massive randomization of a single EM-based algorithm and the other one relies on three different hierarchical, EM-based, and an iterative refinement algorithm with an information-theoretical heuristic. Comparative results are provided with other document clustering algorithms. Zhang, Cheng, Zhang, Chen and Fang (2008) propose a Genetic Algorithm for document clustering and show the improved results in the final clustering over the input clusterings. Other related work is given in Xu et al. (2008), Sevillano et al. (2006).

1.3 THESIS OVERVIEW

This thesis proposes three novel efficient and scalable methods for combining multiple clusterings. These methods are described in the succeeding chapters where accuracy and efficiency of these methods are also demonstrated.

Chapter 1 provides preliminaries of Clustering and Combining Multiple Clusterings, together with a detailed literature survey, for better understanding of concepts in the rest of this thesis.

The following three chapters present the three methods. Chapter 2 presents a novel binary method for fast computation of an objective function. This function measures total inter and intra class similarities of a clustering efficiently to determine its quality based on object co-associations. Time and space complexity of this task is conventionally high. However, the proposed method improves performance of the consensus solution by consuming incomparably less memory and CPU time.

Chapter 3 presents a novel solution for combining multiple clusterings. Our contributions are a novel method for combining a collection of clusterings into a final clustering which is based on cliques, and a novel output-sensitive clique finding algorithm which works on larger graphs and produces output in a short amount of time. Extensive experimental studies on real and artificial data sets demonstrate the effectiveness of our methods.

In chapter 4, a parallel algorithm is proposed to calculate approximate distances of objects by utilizing randomly obtained weak clusterings in shared and distributed memory parallel computing environments.

Finally, chapter 5 summarizes the arguments of this thesis and mentions some open issues in Combining Multiple Clusterings research.

2. FASTFIT: AN EFFICIENT OBJECTIVE FUNCTION FOR EVALUATING CLUSTERING QUALITY

In this chapter, we introduce a novel binary method for fast computation of an objective function. This function measures total inter and intra class similarities of a clustering efficiently to determine its quality based on object co-associations. Thus, it can be used for combining multiple clusterings using optimization-based consensus solutions like evolutionary algorithms. Compared to the conventional technique, our method improves performance of the consensus solution by consuming incomparably less memory and CPU time by considerably reducing space and time complexities respectively. Experimental test results also demonstrate the effectiveness of our new method.

2.1 INTRODUCTION

Some solutions for combining multiple clusterings are optimization-based. These include evolutionary algorithms such as in Cristofor and Simovici (2002), Mohammadi et al. (2008), Zhang, Cheng, Zhang, Chen and Fang (2008), Simulated Annealing as in Filkov and Skiena (2004), Tumer and Agogino (2008). A greedy, info-theoretical optimization method is presented in Strehl and Ghosh (2002). And, Azimi et al. (2009), Yang et al. (2006) present methods based on Swarm Intelligence such as Ant Colony Optimization. All these methods operate on an objective function.

In this chapter, we present a novel and efficient method for computation of a similarity-based objective function. The chapter is structured as follows; Section 2.2 provides preliminary definitions together with some explanations for combining multiple clusterings for clarity. Section 2.3 explains the notion of intra-cluster and inter-cluster similarities. Section 2.4 presents our new binary method for computing intra-cluster and inter-cluster similarities. Experimental Results section provides comparison of our new method with the conventional technique for varying size data sets. In the final section, we present conclusions and future work.

2.2 PRELIMINARIES

Let $\mathcal{D}$ be a data set. A clustering (partition) of $\mathcal{D}$, $\pi(\mathcal{D})$, can be stated as follows;

$$\pi(\mathcal{D}) = \{C_1, C_2, \ldots, C_{|\pi(\mathcal{D})|}\},$$

where C_i is a cluster (block) of $\pi(\mathcal{D})$, $1 \leq i \leq |\pi(\mathcal{D})|$, and

$$\mathcal{D} = \bigcup_{i=1}^{|\pi(\mathcal{D})|} C_i$$

Given a set of clusterings, $\Pi(\mathcal{D}) = \{\pi_1(\mathcal{D}), \pi_2(\mathcal{D}), \ldots, \pi_{|\Pi(\mathcal{D})|}(\mathcal{D})\}$, the problem of combining multiple clusterings is defined as finding a new clustering, $\pi^{\star}(\mathcal{D}) = \{C_1^{\star}, C_2^{\star}, \ldots, C_{|\pi^{\star}(\mathcal{D})|}^{\star}\}$, by using the information provided by $\Pi(\mathcal{D})$. An *objective function* ϕ,

$$\forall i(\phi(\pi^{\star}(\mathcal{D})) \geq \phi(\pi_i(\mathcal{D}))), 1 \leq i \leq |\Pi(\mathcal{D})| \tag{2.1}$$

is used for determining quality of the final clustering $\pi^{\star}(\mathcal{D})$. Exhaustively searching all the possible clusterings for finding *the best* clustering is not an option, since there are approximately $\frac{|\pi^{\star}(\mathcal{D})|^{|\mathcal{D}|}}{|\pi^{\star}(\mathcal{D})|!}$ possible clusterings for every $|\pi^{\star}(\mathcal{D})|$ and $(|\pi^{\star}(\mathcal{D})| \ll |\mathcal{D}|)$ corresponding to the Stirling number of the second kind (see Jain and Dubes (1988)).

Simply put, a *consensus function* is a function which combines multiple clusterings into a single clustering in a sophisticated way so that Formula (2.1) holds. In Fred and Jain (2005), proposed consensus function is based on a co-association measure (simultaneous occurrences) defined by

$$coassoc(d_i, d_j) = votes_{ij}/|\Pi(\mathcal{D})|, \tag{2.2}$$

where $|\Pi(\mathcal{D})|$ is the number of clusterings, $votes_{ij}$ is the number co-occurrences of data objects d_i and d_j in a cluster, such that $d_i \in \mathcal{D}$ and $d_j \in \mathcal{D}$. In this approach, the co-occurrences are held in a $|\mathcal{D}| \times |\mathcal{D}|$ co-association matrix. In general, it is costly to construct, store and populate this matrix for large data sets, because of its $O(|\mathcal{D}|^2)$

complexity. Besides, searching cluster similarity continuously at the object level is a computationally expensive task.

In Filkov and Skiena (2004) and Cristofor and Simovici (2002), consensus functions based on median partition approach have been proposed. This approach searches differences between the clusterings, by working on a coarser level. At another direction, in Strehl and Ghosh (2002), consensus functions based on hypergraphs have been proposed. In this technique, a hyperedge represents a cluster, and a hypergraph represents a clustering.

In this chapter, we focus on a specific objective function which can be used by evolutionary consensus solutions and optimize its time and space complexities. This objective function is presented in the next section.

2.3 INTER AND INTRA CLUSTER SIMILARITIES

Intra-cluster similarity measures how near the data objects are in a cluster. For a clustering, $\pi(\mathcal{D}) = \{C_1, C_2, \ldots, C_{|\pi(\mathcal{D})|}\}$, intra-cluster similarity is measured as follows:

$$ICS(\pi(\mathcal{D})) = \sum_{i=1}^{|\pi(\mathcal{D})|} \frac{1}{|C_i|^2} \sum_{d,d' \in C_i} similarity(d, d') \tag{2.3}$$

For the same clustering, inter-cluster similarity is defined as follows:

$$ECS(\pi(\mathcal{D})) = \sum_{i=1}^{|\pi(\mathcal{D})|} \sum_{j=i+1}^{|\pi(\mathcal{D})|} \frac{1}{|C_i||C_j|} \sum_{d \in C_i, d' \in C_j} similarity(d, d') \tag{2.4}$$

$similarity(d, d')$ is the number of times that objects d and d' are assigned to the same clusters, which is computed from pre-existing multiple clusterings. Finally, ICS and ECS measures can be combined to have following fitness function:

$$\phi(\pi(\mathcal{D})) = k_1.ICS(\pi(\mathcal{D})) + k_2.ECS(\pi(\mathcal{D})) \tag{2.5}$$

Final clustering is expected to have compact, and close clusters, therefore for a good clustering Formula (2.3) should provide large values. Similarly, separated (isolated) clusters are expected in a good clustering, therefore Formula (2.4) shall supply small values. k_1, and k_2 parameters in Formula (2.5) are user defined values, which satisfy $k_1 > 0$ and $k_2 < 0$.

Mohammadi et al. (2008) uses a special case of this objective function to combine multiple clusterings using an evolutionary algorithm and report successful results on small data sets. Inspection of Formula (2.5) reveals that its time complexity is quadratic with respect to the number of objects in the data set, $O(|\mathcal{D}|^2)$. This complexity is due to pairwise similarity calculations performed in Formulas (2.3) and (2.4). For large data sets, holding a similarity matrix is also prohibitive due to its space complexity. Our aim is to reduce these complexities, and compute Formula (2.5) faster using binary methods.

It is important to note that in combining multiple clustering problem, $similarity(d, d')$ is not computed from the data set $\mathcal{D}$. $similarity(d, d')$ in Formulas (2.3) and (2.4) refer to number of co-occurrences of object d, and d' in the same cluster (related to Formula (2.2)). Therefore, it is crucial to utilize the information provided by the pre-existing multiple clusterings.

2.4 FAST COMPUTATION OF SIMILARITIES

We represent each cluster with a bit vector. Existence of an object in a cluster is shown by a 1, similarly absence of an object is captured by a 0. Each cluster representation is as large as the size of the database, $|\mathcal{D}|$. Three clusterings, each having four clusters are shown in Figure 2.1. An example of a cluster is shown below: cluster $\mathbf{C_{11}}$ has data objects d_1, d_2, and d_7.

$\mathbf{C_{11}}$

1	1	0	0	0	0	1	0

In order to compute Formulas (2.3) and (2.4), each cluster has to be examined for pairwise objects, and corresponding entries in the co-association matrix has to be updated. For example, cluster $\mathbf{C_{11}}$ increments following object pairs in the co-association matrix: (d_1, d_1), (d_1, d_2), (d_1, d_7), (d_2, d_2), (d_2, d_7), and (d_7, d_7). Because of the pairwise increment nature, this computation has quadratic time complexity. We present our novel method below for reducing the time complexity of this operation.

Let $\Pi(\mathcal{D})$ be multiple clusterings, and $\pi^\star(\mathcal{D})$ be a new final clustering for a data set $\mathcal{D}$. Following, we define intra-cluster, and inter-cluster similarities of $\pi^\star(\mathcal{D})$. Intra-cluster similarity of $\pi^\star(\mathcal{D})$ is shown in Formula (2.6)

$$ICS_\Pi(\pi^\star(\mathcal{D})) = \sum_{k=1}^{|\pi^\star(\mathcal{D})|} \frac{1}{|C_k^\star|^2} \sum_{i=1}^{|\Pi|} \sum_{j=1}^{|\pi_i|} \binom{|C_k^\star \wedge C_{ij}|}{2} \tag{2.6}$$

And, Formula (2.7) shows the inter-cluster similarity of $\pi^\star(\mathcal{D})$.

$$\begin{aligned} ECS_\Pi(\pi^\star(\mathcal{D})) = \sum_{k=1}^{|\pi^\star(\mathcal{D})|} \sum_{l=k+1}^{|\pi^\star(\mathcal{D})|} \frac{1}{|C_k^\star||C_l^\star|} \sum_{i=1}^{|\Pi|} \sum_{j=1}^{|\pi_i|} & \binom{|(C_k^\star \vee C_l^\star) \wedge C_{ij}|}{2} \\ & - \binom{|C_k^\star \wedge C_{ij}|}{2} - \binom{|C_l^\star \wedge C_{ij}|}{2} \end{aligned} \tag{2.7}$$

Finally, the objective function is described as:

$$\phi_\Pi(\pi^\star(\mathcal{D})) = k_1.ICS_\Pi(\pi^\star(\mathcal{D})) + k_2.ECS_\Pi(\pi^\star(\mathcal{D})) \tag{2.8}$$

$ICS_\Pi(\pi^\star(\mathcal{D}))$ is computed as follows; every cluster of $\pi^\star(\mathcal{D})$ is logically ANDed with every cluster in Π in order to find pairwise co-occurrences of the objects in the same cluster. Similarly, when computing $ECS_\Pi(\pi^\star(\mathcal{D}))$ every cluster pair in $\pi^\star(\mathcal{D})$ is logically ORed in order to find all pairs of objects, this result is ANDed with every cluster in Π in order to find pairwise co-occurrences of objects. So far we obtained the pairwise co-occurrences of objects in the same clusters and in two different clusters. By subtracting pairwise co-occurrences of the objects in the same clusters (last two components in Formula (2.7)), we obtain pairwise co-occurrences of objects in different clusters as shown in the formula. When computed on the same set of input clusterings, Π, Formula (2.6) is equivalent to Formula (2.3) and Formula (2.7) is equivalent to Formula (2.4). Although equivalent formulas yield the same result, it is very important to note that Formulas (2.6) and (2.7) are computed in cluster level, not in object level for each pairwise object. Therefore, Formulas (2.6) and (2.7) are very efficient when compared to the Formulas (2.3) and (2.4). As a result, time complexity of Formula (2.8) is reduced to $O(|\pi^{\star}(\mathcal{D})||\Pi|_{tc} + |\pi^{\star}(\mathcal{D})|^2|\Pi|_{tc})$, where $|\Pi|_{tc}$ represents total number of clusters in $|\Pi|$ (e.g. $|\Pi|_{tc}$ in Figure 2.1 is 9) and $|\Pi|_{tc} << |\mathcal{D}|$, and $|\pi^{\star}(\mathcal{D})|^2 << |\mathcal{D}|$. In the next section effectiveness of our new technique is demonstrated.

$\mathcal{D}$		$\mathbf{d_1}$	$\mathbf{d_2}$	$\mathbf{d_3}$	$\mathbf{d_4}$	$\mathbf{d_5}$	$\mathbf{d_6}$	$\mathbf{d_7}$	$\mathbf{d_8}$
$\pi_1(\mathcal{D})$	$\mathbf{C_{11}}$	1	1	0	0	0	0	1	0
	$\mathbf{C_{12}}$	0	0	1	1	1	0	0	0
	$\mathbf{C_{13}}$	0	0	0	0	0	1	0	1
$\pi_2(\mathcal{D})$	$\mathbf{C_{21}}$	1	0	1	1	0	0	0	0
	$\mathbf{C_{22}}$	0	1	0	0	1	1	1	1
$\pi_3(\mathcal{D})$	$\mathbf{C_{31}}$	1	1	0	0	0	0	0	0
	$\mathbf{C_{32}}$	0	0	1	1	0	0	0	0
	$\mathbf{C_{33}}$	0	0	0	0	1	1	0	0
	$\mathbf{C_{34}}$	0	0	0	0	0	0	1	1

Figure 2.1: Binary representation of multiple clusterings, $\Pi(\mathcal{D})$

		$\mathbf{d_1}$	$\mathbf{d_2}$	$\mathbf{d_3}$	$\mathbf{d_4}$	$\mathbf{d_5}$	$\mathbf{d_6}$	$\mathbf{d_7}$	$\mathbf{d_8}$
$\pi^{\star}(\mathcal{D})$	$C_1^{\star}$	1	1	1	1	0	0	0	0
	$C_2^{\star}$	0	0	0	0	1	1	1	1

Figure 2.2: Binary representation of $\pi^{\star}(\mathcal{D})$

Example 2.4.1. *Let us compute intra-cluster similarity, $ICS_{\Pi}(\pi^{\star}(\mathcal{D}))$, of $\pi^{\star}(\mathcal{D})$ of Figure 2.2 using multiple clusterings shown in Figure 2.1 by using the Formula (2.6)*

$$ICS_{\Pi}(\pi^{\star}(\mathcal{D})) = \frac{1}{4^2}\left(\binom{2}{2} + \binom{2}{2} + \ldots + \binom{2}{2}\right)$$
$$+\frac{1}{4^2}\left(\binom{1}{2} + \binom{1}{2} + \ldots + \binom{2}{2}\right)$$

In a similar manner, Formula (2.7) can be used for computing $ECS_{\Pi}(\pi^{\star}(\mathcal{D}))$.

2.4.1 Generalization of the Method

Formula (2.7) is valid for non-overlapping clusters in a clustering. However, it can be generalized for overlapping clusters as follows:

$$\mathcal{ECS}_{\Pi}(\pi^{\star}(\mathcal{D})) = \sum_{k=1}^{|\pi^{\star}(\mathcal{D})|} \sum_{l=k+1}^{|\pi^{\star}(\mathcal{D})|} \frac{1}{|C_k^{\star}||C_l^{\star}|}$$

$$\sum_{i=1}^{|\Pi|}\sum_{j=1}^{|\pi_i|} \binom{|(C_k^{\star} \vee C_l^{\star}) \wedge C_{ij}|}{2} - \binom{|(C_k^{\star} \wedge \overline{C_l^{\star}}) \wedge C_{ij}|}{2}$$

$$-\binom{|(\overline{C_k^{\star}} \wedge C_l^{\star}) \wedge C_{ij}|}{2} + \binom{|(C_k^{\star} \wedge C_l^{\star}) \wedge C_{ij}|}{2}$$

$$+ (|\Pi||C_k^{\star} \wedge C_l^{\star}|) \tag{2.9}$$

Above, $C_k^{\star}$ (also $C_l^{\star}$) represents a final cluster, C_{ij} represents a pre-existing cluster, and $\overline{C_k^{\star}}$ represents complement of $C_k^{\star}$.

It should be noted that $\mathcal{ICS}(\pi^{\star}(\mathcal{D}))$ is the same as $ICS_{\Pi}(\pi^{\star}(\mathcal{D}))$ for non-overlapping clusters. Therefore, generalized form of the objective function, FastFit, is shown in (2.10).

$$FastFit(\pi^{\star}(\mathcal{D})) = k_1.\mathcal{ICS}(\pi^{\star}(\mathcal{D})) + k_2.\mathcal{ECS}(\pi^{\star}(\mathcal{D})) \tag{2.10}$$

Further explanation of (2.9): It is a generalization of (2.7) used for both overlapping and non-overlapping final clusters. Term 1 refers to $similarity(d, d')$ for every pair of objects in a pair of final clusters: $C_k^{\star}$, $C_l^{\star}$. Term 2 subtracts intra-cluster contribution of $C_k^{\star} - C_l^{\star}$. Similarly, Term 3 subtracts intra-cluster contribution of $C_l^{\star} - C_k^{\star}$. Term 4 adds duality between pairs in overlapping region : Notice there is an arrow from 1 to 7, and 7 to 1 in Figure 2.3(c). The last term is needed in order to include $similarity(d, d)$ contributions in overlapping region: See the self loops for objects 1 and 7 in the same figure.

$$\mathcal{ECS}_{\Pi}(\pi^{\star}(\mathcal{D})) = \sum_{k=1}^{|\pi^{\star}(\mathcal{D})|} \sum_{l=k+1}^{|\pi^{\star}(\mathcal{D})|} \frac{1}{|C_k^{\star}||C_l^{\star}|}$$

$$\sum_{i=1}^{|\Pi|}\sum_{j=1}^{|\pi_i|} \underbrace{\binom{|(C_k^{\star} \vee C_l^{\star}) \wedge C_{ij}|}{2}}_{\mathtt{1}} - \underbrace{\binom{|(C_k^{\star} \wedge \overline{C_l^{\star}}) \wedge C_{ij}|}{2}}_{\mathtt{2}}$$

$$- \underbrace{\binom{|(\overline{C_k^\star} \wedge C_l^\star) \wedge C_{ij}|}{2}}_{3} + \underbrace{\binom{|(C_k^\star \wedge C_l^\star) \wedge C_{ij}|}{2}}_{4}$$

$$+ \underbrace{(|\Pi||C_k^\star \wedge C_l^\star|)}_{5}$$

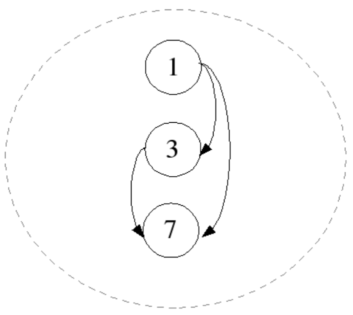
2.5 EXPERIMENTAL RESULTS

Our experimental setup is a PC having 2.0 GHz processor with 2 GB main memory. No parallel processing was done. Our choice of implementation language is Java, which provides built-in support for bit vectors, and operations on bit vectors.

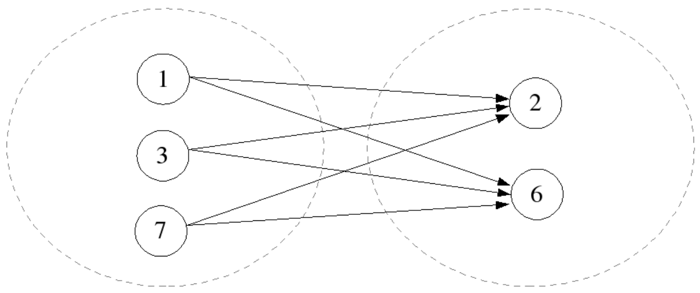
We have conducted experiments on varying size synthetic and real Data Sets. Details of the Data Sets and their Multiple Clusterings are given in Table 2.1 where, besides the convention used in this chapter, $|\pi^o(\mathcal{D})|$ refers to number of inherent original classes in a data set, $|\pi(\mathcal{D})|$ refers to number of clusters in each clustering in the set of Multiple clusterings, and $|\pi^\star(\mathcal{D})|$ is the number of clusters in a tested Final Clustering.

IRIS, GLASSIDE, IMAGESEG and SIGNFORM are data sets from UCI repository Asuncion and Newman (2007). SYNTH1, SYNTH2 and SYNTH3 are synthetically generated up to 20,000 data objects. Larger data sets SYN4C100 and SYN4C1M are also synthetically generated with 100K and 1M data objects respectively and contain a mixture of 4-Gaussians each.

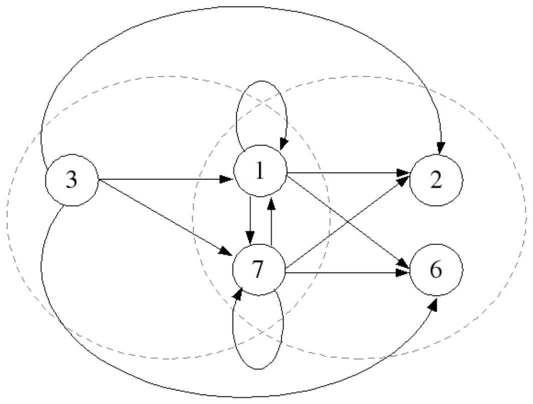
Table 2.1 displays comparative execution time results between FastFit and the conventional matrix operations. In all experiments, FastFit obviously outperforms conventional method. With large data sets, a comparison is not even possible, because a $|\mathcal{D}| \times |\mathcal{D}|$ matrix does not fit in the main memory. In Figure 2.4 execution time results of a data set having 5,000 objects is shown for different $|\Pi(\mathcal{D})|$s from 10 to 100. Clearly, FastFit is superior to the conventional method. We provided the time axis in logarithmic scale, because our execution time results are very small when compared to the conventional method. Figures 2.5, and 2.6 represent similar results.



(a) Intra Cluster Similarity



(b) Graph Showing Similarity of Two Non-Overlapping Clusters



(c) Pseudograph Showing Similarity of Two Overlapping Clusters

Figure 2.3: Graphical Representation of Cluster Similarities

Table 2.2 compares memory consumption of FastFit with the conventional method. FastFit uses incomparably low memory, which enables working with very large data sets. Although shown in the table for completeness, in practice, it is not possible to apply conventional method to Data Sets with 100K and 1M objects. Memory consumption results are also shown Figure in 2.7.

Table 2.1: Experimental Setup and Run Times

Data set	$\lvert\mathcal{D}\rvert \times \lvert Attribs\rvert$	$\lvert\pi^o(\mathcal{D})\rvert$	$\lvert\Pi\rvert$	$\lvert\pi(\mathcal{D})\rvert$	$\lvert\pi^\star(\mathcal{D})\rvert$	Avg. Run-Time FastFit (ms)		Avg. Run-Time Conventional (ms)		Matrix Initialization (ms)
						ICS	ECS	ICS	ECS	
IRIS	150×4	3	10	2 - 6	3	0.35	0.38	0.78	0.82	2.68
GLASSIDE	214×9	6	10	4 - 10	6	0.96	2.34	3.06	10.02	7.21
IMAGESEG	2,310×19	7	10	4 - 10	7	4.51	32.39	48.25	85.69	209.39
SIGNFORM	5K×21	3	10	2 - 6	3	3.33	11.40	109.13	158.86	1,523.50
SYNTH1	5K×20	n/a	10	2 - 6	4	1.54	2.45	156.12	437.34	1,085.42
SYNTH2	10K×20	n/a	10	2 - 6	4	1.93	4.58	595.18	1,764.97	7,088.23
SYNTH3	20K×20	n/a	10	2 - 6	4	2.87	9.23	2,296.57	6,986.60	19,382.31
SYN4C100	100K×2	4	10	2 - 6	4	25.95	41.66	Out of Memory		
SYN4C1M	1M×2	4	10	2 - 6	4	45.48	97.60	Out of Memory		

2.6 CONCLUSIONS AND FUTURE WORK

In this chapter, we presented a novel binary method for computing intra-cluster and inter-cluster similarities. By representing each cluster using a bit vector, we utilize fast operations and low memory requirements of binary operations. Our method is especially useful as an objective function for optimization-based consensus solutions like evolutionary algorithms for combining multiple clusterings.

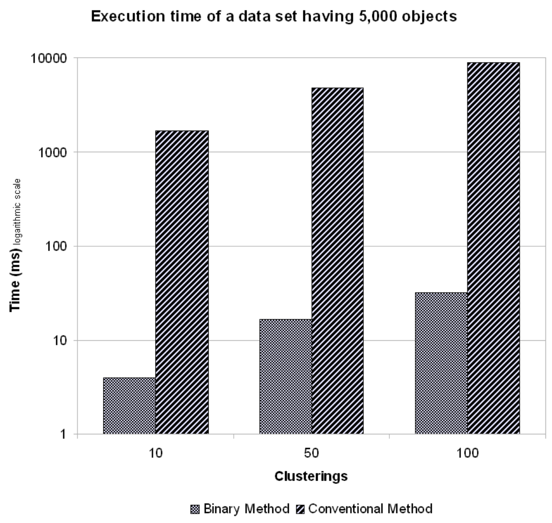


Figure 2.4: Execution time on SYNTH1 (5,000 objects)

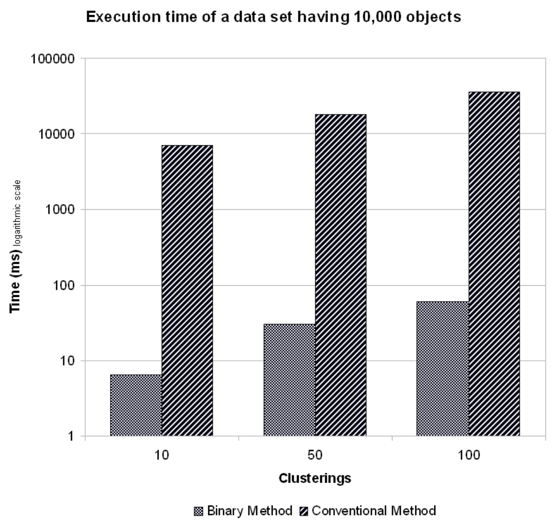


Figure 2.5: Execution time on SYNTH2 (10,000 objects)

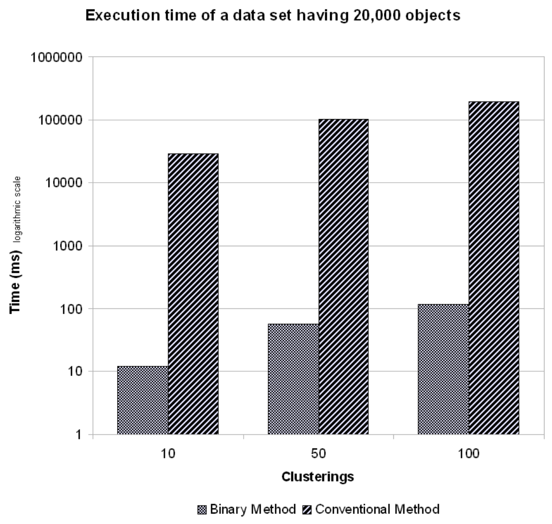


Figure 2.6: Execution time on SYNTH3 (20,000 objects)

Experimental results on varying size synthetic and real data sets, including large data sets, demonstrate the effectiveness of our method. Our method is superior to the conventional technique, in that, it is fast and efficient, and it scales to large data sets and uses incomparably less memory.

By using the objective function here, we will investigate designing a novel method for combining multiple clusterings as a future work. Evolutionary based techniques can use our consensus function presented here as a fitness function. Generalizing our technique for broader use will also be a future work.

Table 2.2: Memory Consumption

$\lvert\mathcal{D}\rvert$	$\lvert\Pi\rvert$	$\lvert\pi(\mathcal{D})\rvert$	Memory Consumption	
			FastFit (MB)	Conventional (MB)
5,000	20	2 - 6	0.04	100
10,000	20	2 - 6	0.08	400
20,000	20	2 - 6	0.16	1,600
40,000	20	2 - 6	0.32	6,400
100,000	20	2 - 6	0.80	40,000
1,000,000	20	2 - 6	8.00	4M

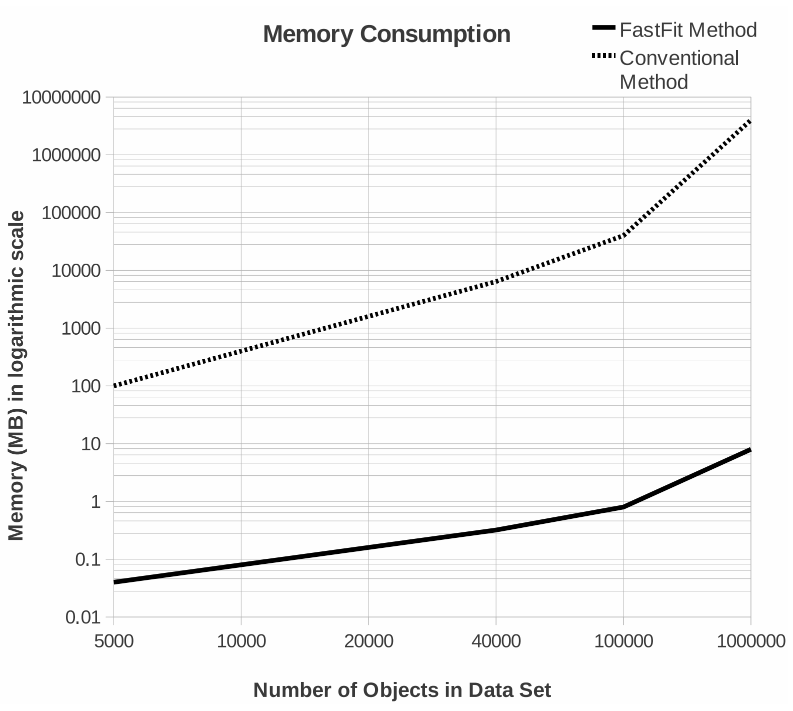


Figure 2.7: Comparison of Memory Consumption (Average $\lvert\pi(\mathcal{D})\rvert$ = 4)

3. CLICOM: CLIQUES FOR COMBINING MULTIPLE CLUSTERINGS

Finding natural groupings of a data set is a hard task as attested by hundreds of clustering algorithms in the literature. Each clustering technique makes some assumptions about the underlying data set. If the assumptions hold, good clusterings can be expected. It is hard, in some cases impossible, to satisfy all the assumptions. Therefore, it may be beneficial to apply different clustering methods on the same data set, or the same method with varying input parameters or both. Then, the obtained clusterings can be combined into a final clustering having better overall quality. Our contributions are a novel method for combining a collection of clusterings into a final clustering which is based on cliques, and a novel output-sensitive clique finding algorithm which works on larger graphs and produces output in a short amount of time. Extensive experimental studies on real and artificial data sets demonstrate the effectiveness of our methods.

3.1 INTRODUCTION

We refer the interested reader to Chapter 1 for an overview of Data Clustering and Combining Multiple Clusterings research.

In this chapter, we present a graph-based algorithm for combining multiple clusterings which is based on the idea of *maximally complete subgraphs*-also known as *cliques*. Cliques form very strong clusters, since in a clique each vertex is connected to all other vertices. In the following sections, we show that utilizing cliques for combining multiple clusterings is effective and practical. We also present an algorithm for finding a substantial subset of all the cliques quickly, since finding all the cliques in a large graph may be computationally overwhelming.

The chapter is organized as follows; Section 3.2 introduces the preliminaries: problem definition and important concepts. Related work is presented in Section 3.3. Section 3.4 describes the main algorithm CLICOM. In section 3.5, we discuss advantages and disadvantages of CLICOM. Section 3.6 presents a novel output-sensitive clique finding method, which is integrated in the main algorithm. Experimental evaluations are presented in Section 3.7. Finally, Section 3.8 concludes the chapter.

3.2 PRELIMINARIES

In this section, combining multiple clusterings, graph-based clustering, and cliques are explained.

3.2.1 Clusterings and Co-associations

Definitions related to multiple clusterings and object co-associations have already been provided in Section 2.2. For completeness of this chapter, we restate some of the definitions here in brief.

Let $\mathcal{D}$ be a data set. A clustering of $\mathcal{D}$ is shown as

$$\pi(\mathcal{D}) = \{C_1, C_2, \ldots, C_{|\pi(\mathcal{D})|}\},$$

where C_i is a cluster in $\pi(\mathcal{D})$, $1 \leq i \leq |\pi(\mathcal{D})|$, and

$$\mathcal{D} = \bigcup_{i=1}^{|\pi(\mathcal{D})|} C_i$$

Figure 3.1 represents a collection of multiple clusterings in binary format; $\Pi(\mathcal{D}) = \{\pi_1(\mathcal{D}), \pi_2(\mathcal{D}), \ldots, \pi_{|\Pi|}(\mathcal{D})\}$ with $|\Pi(\mathcal{D})|$ clusterings and $|\Pi|_{tc}$ clusters in total. The problem of combining multiple clusterings refers to producing a new clustering $\pi^\star(\mathcal{D}) = \{C_1^\star, C_2^\star, \ldots, C^\star_{\pi^\star|(\mathcal{D})|}\}$, which has better overall quality and uses the information provided by $\Pi(\mathcal{D})$. A validity function ϕ is used to measure quality of a clustering.

$$\forall i(\phi(\pi^\star(\mathcal{D})) \geq \phi(\pi_i(\mathcal{D}))), 1 \leq i \leq |\Pi(\mathcal{D})| \tag{3.1}$$

Many consensus functions, which produce final output clusterings, have been proposed in the literature such as Fred and Jain (2005), Strehl and Ghosh (2002). In Fred and Jain (2005), proposed consensus function is based on a natural co-association measure in which similarity of objects in $\mathcal{D}$ is based on the number of times they coexist in the same clusters. An efficient method for calculating inter and intra cluster similarities of a cluster which is based on co-association object pairs is given in Mimaroglu and Yagci (2009).

$\mathcal{D}$		$\mathbf{d_1}$	$\mathbf{d_2}$	$\mathbf{d_3}$	$\mathbf{d_4}$	$\mathbf{d_5}$	$\mathbf{d_6}$	$\mathbf{d_7}$	$\mathbf{d_8}$
$\pi_1(\mathcal{D})$	$\mathbf{C_{11}}$	1	1	0	0	0	0	0	0
	$\mathbf{C_{12}}$	0	0	1	1	1	0	0	0
	$\mathbf{C_{13}}$	0	0	0	0	0	1	1	1
$\pi_2(\mathcal{D})$	$\mathbf{C_{21}}$	1	1	1	1	0	0	0	0
	$\mathbf{C_{22}}$	0	0	0	0	1	1	1	1
$\pi_3(\mathcal{D})$	$\mathbf{C_{31}}$	1	1	0	0	0	0	0	0
	$\mathbf{C_{32}}$	0	0	1	1	0	0	0	0
	$\mathbf{C_{33}}$	0	0	0	0	1	1	0	0
	$\mathbf{C_{34}}$	0	0	0	0	0	0	1	1

Figure 3.1: A Collection of Clusterings $\Pi(\mathcal{D})$, $|\mathcal{D}| = 8$, $|\Pi(\mathcal{D})| = 3$ and $|\Pi|_{tc} = 9$

3.2.2 Graph-based Clustering

Graph-based clustering methods work on a graph representation $G = (V, E)$ of a data set Tan et al. (2005). In graph representation, a vertex represents an object, or some higher level information of the data such as a cluster or a cluster center. Proximities between vertices are captured by edges and edge labels. Graph-based clustering methods start with creating a $|V| \times |V|$ proximity matrix between each pair of vertices, which holds edge labels. A sparsification of the graph can be performed for avoiding noise and improving cluster quality e.g. by using a threshold or by keeping only k-nearest neighbors of each vertex. Graph-theoretical techniques are applied to partition this final graph in a meaningful way.

3.2.3 Cliques

A clique is a maximally complete subgraph of an undirected graph $G = (V, E)$ Moon and Moser (1965). A subgraph of G is a graph $G' = (V', E')$ where $V' \subseteq V$ and $E' \subseteq E$. G' is complete, if for every $u, v \in V'$, $(u, v) \in E'$. The set of vertices V' forms a clique if it is maximally complete, i.e. there is no set such that $V'' \subseteq V$, $V' \subset V''$ and V'' is complete.

There can be at most $\alpha.3^{|V|/3}$ cliques Moon and Moser (1965) in a graph G, and finding all the cliques of a graph has exponential time complexity. In the literature, several algorithms Akkoyunlu (1973), Bron and Kerbosch (1973), Tomita et al. (2006) have been proposed to prune the search space without relaxing the original clique definition, thus enabling efficient time and space complexities. Output-sensitive techniques by limiting the output size (number of cliques) achieve better run-time. We present a novel output-sensitive clique finding algorithm in Section 3.6.

Bron and Kerbosch (1973) propose a family of seminal algorithms which consists of two recursive backtracking algorithms that use branch-and-bound technique to prune branches of recursion tree which cannot lead to a clique. These algorithms and their improved versions, such as the one proposed by Samudrala and Moult (1998), are used widely. We use the same implementation as well. In a more recent and influential work which is presented in Tomita et al. (2006), a depth-first search algorithm is proposed which utilizes pruning methods of Bron and Kerbosch (1973). Worst case run-time complexity of the proposed algorithm is reported to be $O(3^{|V|/3})$.

In the following section, we are interested in finding all the cliques of a graph. However, on dense or very large graphs, we have to use our novel output-sensitive clique algorithm in order to generate good quality final clusterings in practical run-times. In graph-based clustering, for our purposes, it is possible to tune the granularity of the graph: In finer granularity approach each data point is a vertex, in coarser granularity approach each (input) cluster is a vertex. Finer granularity approach is very rarely practical because of its vertex size and density. On the other hand, coarser granularity approach is generally practical due to its small size, i.e. $|\Pi|_{tc} \ll |\mathcal{D}|$. For both cases, required graphs are obtained by processing a collection of clusterings as shown in Figure 3.1.

Object-wise cliques

An object-wise similarity graph can be represented by a co-association matrix, M Fred and Jain (2005). M_{ij} is the number of times that objects d_i and d_j are assigned to the same cluster. This is also known as *Evidence Accumulation* Fred and Jain (2005) and can easily be obtained from Figure 3.1 by counting 1s for each object pair. Our main algorithm, CLICOM, can also run on this graph which has as many vertices as $|\mathcal{D}|$.

Cluster-wise cliques

In a cluster-wise similarity graph, each vertex represents an input cluster. Cluster-wise similarities are computed using Formula (3.2) on the input clusterings as shown in Figure 3.1. Other similarity measures, such as Jaccard similarity can also be used. It is very important to note that number of total clusters is much smaller than the data set size, i.e. $|\Pi|_{tc} \ll |\mathcal{D}|$. In most cases, cliques on cluster-wise similarity graph can be computed quickly.

3.3 RELATED WORK

In this section, we provide an overview of related and influential work for combining multiple clusterings.

CSPA (Cluster-Based Similarity Partitioning Algorithm), which is introduced in Strehl and Ghosh (2002), is based on a co-association matrix, and METIS, which is a software package for partitioning unstructured graphs Karypis and Kumar (1998b,a). CSPA is shown in Algorithm 3.

```
Input: $\Pi(\mathcal{D})$: Multiple Clusterings, $k$: Number of Clusters In the Final Clustering
Output: $\pi^*(\mathcal{D})$: Final Clustering
Compute $SM$, $|\mathcal{D}| \times |\mathcal{D}|$ co-association matrix , using $\Pi(\mathcal{D})$ ;
$\pi^*(\mathcal{D}) = \mathbf{METIS}(SM, k)$ ;
return $\pi^\star(\mathcal{D})$;
```

Algorithm 3: Cluster-Based Similarity Partitioning Algorithm **CSPA**

HGPA(Hypergraph Partitioning Algorithm) is introduced in Strehl and Ghosh (2002) as well. Multiple clusterings construct a hypergraph where each object is a vertex, and each cluster is a hyperedge. Main idea is to have k unconnected components of the hypergraph by using HMETIS Karypis, Aggarwal, Kumar and Shekhar (1999). Combining multiple clusterings problem is formulated as partitioning the hypergraph by cutting a minimal number of hyperedges. A set of hyperedges are removed and k unconnected components are obtained, which provides the final clustering.

In Meta-Clustering Algorithm (MCLA) Strehl and Ghosh (2002), each cluster is represented by a hyperedge, like in HGPA. MCLA is composed of the following steps: 1.Constructing the meta-graph, a regular graph labelled with Jaccard similarity between a pair of hyperedges. 2.Partitioning the meta-graph, 3.Computing cluster members. These steps are shown in Algorithm 4.

```
Input: $\Pi(\mathcal{D})$: Multiple Clusterings $k$: Number of Clusters in the Final Clustering
Output: $\pi^*(\mathcal{D})$: Final Clustering
// $G$ is a meta-graph, construct it
$G = (V, E)$ ;
foreach $c \in \Pi(\mathcal{D})$ do
    Add $c$ as a vertex to V;
foreach $v_1 \in V$ do
    foreach $v_2 \in V$ do
        if $v_1 \neq v_2$ then
            // label edge $(v_1, v_2)$ using Jaccard similarity
            $label(v_1, v_2) = \frac{|v_1 \cap v_2|}{|v_1 \cup v_2|}$ ;
$\pi^\star(\mathcal{D}) = \mathbf{METIS}(G, k)$ ;
foreach object $d \in \mathcal{D}$ do
    // modify $\pi^\star(\mathcal{D})$ as follows
    assign $d$ to its most associated cluster in $\pi^\star(\mathcal{D})$
return $\pi^\star(\mathcal{D})$;
```

Algorithm 4: Meta-Clustering Algorithm **MCLA**

Evidence Accumulation (EAC) Fred and Jain (2005) accumulates the evidence in each cluster to form a co-association matrix, SM. Each entry in this matrix, SM_{ij}, is the number of times that data set objects d_i and d_j are assigned to the same clusters. The similarity matrix is provided as input to an agglomerative clustering algorithm, as shown in Algorithm 5.

```
Input: $\Pi(\mathcal{D})$: Multiple Clusterings, $k$: Number of Clusters in the Final Clustering
Output: $\pi^*(\mathcal{D})$: Final Clustering
Compute $SM$, $|\mathcal{D}| \times |\mathcal{D}|$ co-association matrix , using $\Pi(\mathcal{D})$ ;
Run Agglomerative Clustering on $SM$ to construct $\pi^*(\mathcal{D})$;
return $\pi^\star(\mathcal{D})$;
```

Algorithm 5: Evidence Accumulation **EAC**

The data set may be distributed at different sites. In this case, a distributed clustering solution with a final merging of clusters is needed. Hore et al. (2009) proposes two methods, Bipartite Merger (BM) and METIS Merger (MM), for combining distributed clusters. Using cluster centers (prototypes) instead of clusters reduces computation and memory requirements. BM works on several clusterings each having the same number, n, clusters. It groups the centroids according to their similarity and merges them to have a final clustering with n clusters. MM uses METIS, and it is more flexible: clusterings can have different number of clusters. Good results of both BM and MM are reported in Hore et al. (2009).

Some of the methods for combining multiple clusterings are based on evolutionary algorithms such as Cristofor and Simovici (2002), Mohammadi et al. (2008). Some other techniques are based on simulated annealing Filkov and Skiena (2004), Tumer and Agogino (2008). A greedy, info-theoretical optimization method is presented in Strehl and Ghosh (2002). All of these methods operate on an objective function.

3.3.1 Shortcomings of Related Work

CSPA, HGPA, MCLA, and EAC require the number of final clusters in advance. BM and MM represent clusters as centroids. For this reason, we will not consider BM and MM in the experimental evaluations. EAC requires a lot of computations, therefore it is slow on all the data sets. Neither EAC nor CSPA scales well. Although HGPA is very fast, it is not very accurate. Evolutionary methods usually suffer from having long run-times.

3.4 CLICOM

Cliques for **com**bining multiple clustering CLICOM, is shown in Algorithm 6. CLICOM starts to work on a weighted graph, where each cluster in a set of input clusterings are represented by vertices. Pairwise similarities of clusters, computed in line 8, label the graph edges. Graph sparsification is performed with respect to a threshold value, θ, as shown in line 9. Sparsified graph representation, A, is supplied as input to findCliques procedure which implements Bron-Kerbosch algorithm as described in Samudrala and Moult (1998). Both the original cluster-wise similarity graph, and its sparsified version are stored in $|\Pi|_{tc} \times |\Pi|_{tc}$ matrices: M and A.

Given a set of multiple clusterings in binary format $\Pi(\mathcal{D})$, we use Formula (3.2) for calculating the similarity between a pair of clusters (inter cluster similarity as shown in Figure 3.2) $ECS_{\Pi(\mathcal{D})}(C_k, C_l)$, using co-associations of objects, which is based on Mimaroglu and Yagci (2009).

```
Input: $\Pi(\mathcal{D})$: Collection of Clusterings of a Data Set $\mathcal{D}$, $\theta$: Threshold for Sparsification
Output: $\pi^{\star}(\mathcal{D})$: Final Clustering of $\mathcal{D}$
Initialize $allClusters = \emptyset$ ;
foreach $\pi_i \in \Pi(\mathcal{D})$ do
    foreach $C_j \in \pi_i(\mathcal{D})$ do
        $allClusters = allClusters \cup C_j$ ;
Initialize proximity and adjacency matrices, $M$ and $A$ ;
foreach $C_i$ in $allClusters$ do
    foreach $C_j$ in $allClusters$, $i \neq j$ do
        $M_{ij} = ECS_{\Pi(\mathcal{D})}(C_i, C_j)$ ;
        if $M_{ij} >= \theta$ then $A_{ij}$ = 1 else $A_{ij}$ = 0 ;
$SC \leftarrow findCliques(A)$ ;
$B \leftarrow cliquesToClusters(SC, M, allClusters)$ ;
$\pi^{\star}(\mathcal{D}) \leftarrow majorityVoter(B)$ ;
return $\pi^{\star}(\mathcal{D})$
```

Algorithm 6: CLICOM

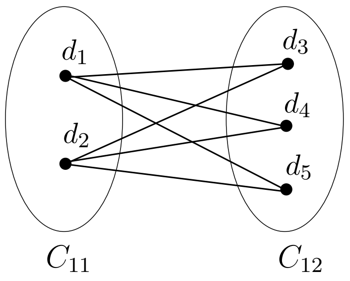


Figure 3.2: Inter Cluster Similarity of a Cluster Pair in $\pi_1(\mathcal{D})$

$$ECS_{\Pi(\mathcal{D})}(C_k, C_l) = \frac{1}{|\Pi||C_k||C_l|} \sum_{i=1}^{|\Pi|} \sum_{j=1}^{|\pi_j|} \binom{|(C_k \vee C_l) \wedge C_{ij}|}{2} - \binom{|(C_k \wedge \overline{C_l}) \wedge C_{ij}|}{2} - \binom{|(\overline{C_k} \wedge C_l) \wedge C_{ij}|}{2} + \binom{|(C_k \wedge C_l) \wedge C_{ij}|}{2} + (|\Pi(\mathcal{D})||C_k \wedge C_l|) \tag{3.2}$$

We use $ECS_{\Pi(\mathcal{D})}$ similarity measure, since it can find cluster-wise similarities based on co-associations of objects rather than e.g. a ratio of syntactically matched objects in the clusters as in Jaccard similarity Tan et al. (2005). Our experimental studies show $ECS_{\Pi(\mathcal{D})}$ is superior to other similarity measures: With proper sparsification, $ECS_{\Pi(\mathcal{D})}$ captures cliques as given in Figure 3.3 where Jaccard measure cannot. This figure shows that when using Jaccard measure, two cliques of size 3 are detected:$\{C_{11}, C_{21}, C_{31}\}$, and

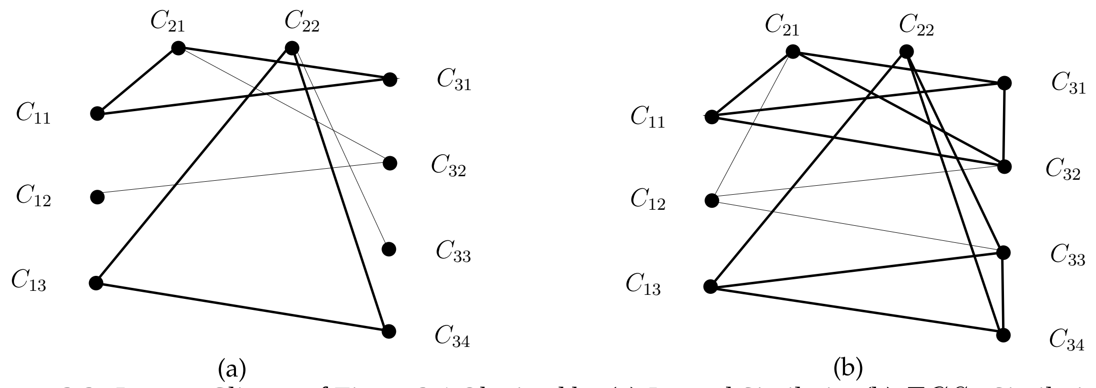


Figure 3.3: Largest Cliques of Figure 3.1 Obtained by (a) Jaccard Similarity (b) ECS_{Π} Similarity

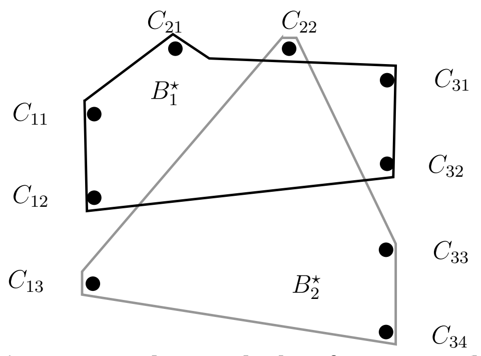


Figure 3.4: Cluster Blocks of Figure 3.3(b)

$\{C_{13}, C_{22}, C_{34}\}$. On the same data set, $ECS_{\Pi(\mathcal{D})}$ similarity reveals two cliques of size 4: $\{C_{11}, C_{21}, C_{31}, C_{32}\}$ and $\{C_{13}, C_{22}, C_{33}, C_{34}\}$.

Cliques returned by the findCliques procedure overlap, cliquesToClusters procedure puts clusters into meaningful cluster blocks by reorganizing them in a non-overlapping manner. The notion of cluster blocks is shown in Figure 3.4, which contains two blocks $B_1^{\star}$, and $B_2^{\star}$. Number of cliques in the similarity graph does not determine the number of clusters in the final clustering; cliquesToClusters detects the number of clusters in the final clustering automatically.

Algorithm 7 begins by ranking cliques using a normalized cumulative pairwise similarity $CECS_{\Pi(\mathcal{D})}$ as given in Formula (3.3). Note that this formula rewards large compact cliques, since such cliques form predominant clusters. Starting with the highest ranking clique, the algorithm iterates through each clique and accepts it as a cluster block which can be expanded with other clusters, only if a certain percentage of its members have not been assigned to any existing block. In experiments, we fixed this ratio to 90% ($\delta = 10\%$). At the end, there may be some vertices that are not assigned to any block. Each such vertex C_i is appended to a block that contains a vertex which is most similar to C_i (see

```
Input: $SC$: Set of Cliques, $M$: Proximity Matrix, $allClusters$: Set of All Clusters, $\delta$ :
        Overlapping Percentage
Output: $B = \{B^\star_1, B^\star_2, ..., B^\star_{|\pi^\star|}\}$: Cluster Blocks
Sort $SC$ by ranking each clique $clique_m \in SC$ in decreasing order with respect to
$CECS_{\Pi(\mathfrak{D})}$ ;
$k \leftarrow 0$ ;
for $m \leftarrow 1$ to $|SC|$ do
    if $\frac{|assigned \cap clique_m|}{|clique_m|} \leq \delta$ then
        $B^\star_k \leftarrow clique_m - assigned$ ;
        $B \leftarrow B \cup \{B^\star_k\}$ ;
        $assigned \leftarrow assigned \cup B^\star_k$ ;
        $k \leftarrow k + 1$ ;
// Cluster unclustered clusters
if $allClusters - assigned \neq \emptyset$ then
    foreach $C_i \in allClusters - assigned$ do
        // Find most similar assigned cluster and return its
           index
        $l \leftarrow max_j(M_{ij}), C_j \in assigned$ ;
        foreach $B^\star_k \in B$ do
            if $C_l \in B^\star_k$ then
                $B^\star_k \leftarrow B^\star_k \cup \{C_i\}$ ;
                $assigned \leftarrow assigned \cup \{C_i\}$ ;
return $B$
```
Algorithm 7: $cliquesToClusters$ Computes Cluster Blocks

lines 9-15).

$$CECS_{\Pi(\mathfrak{D})}(clq_m) = \frac{|clq_m|}{\binom{|clq_m|}{2}} \sum_{i=1}^{|clq_m|} \sum_{j=i+1}^{|clq_m|} ECS_{\Pi(\mathfrak{D})}(C_i, C_j) \tag{3.3}$$

Figure 3.4 illustrates an example: Largest two cliques in Figure 3.3(b) $\{C_{11}, C_{21}, C_{31}, C_{32}\}$ and $\{C_{13}, C_{22}, C_{33}, C_{34}\}$ are picked as blocks, and vertex C_{12} is appended to the first block, since it is most similar to C_{21}.

Final step is to convert the cluster blocks into clusters of data objects, which is carried out by majorityVoter as shown in Algorithm 8. Using cluster blocks and their content, this algorithm counts the number of occurrences of objects as shown in lines 5-8. Lines 9-11 assign each object to a cluster depending on its frequency. Figure 3.5 shows the voting procedure, and Figure 3.6 presents the final clustering produced by CLICOM on the input shown in Figure 3.1.

		$\mathbf{d_1}$	$\mathbf{d_2}$	$\mathbf{d_3}$	$\mathbf{d_4}$	$\mathbf{d_5}$	$\mathbf{d_6}$	$\mathbf{d_7}$	$\mathbf{d_8}$
$B_1^\star$	C_{11}	1	1	0	0	0	0	0	0
	C_{12}	0	0	1	1	1	0	0	0
	C_{21}	1	1	1	1	0	0	0	0
	C_{31}	1	1	0	0	0	0	0	0
	C_{32}	0	0	1	1	0	0	0	0
$B_2^\star$	C_{13}	0	0	0	0	0	1	1	1
	C_{22}	0	0	0	0	1	1	1	1
	C_{33}	0	0	0	0	1	1	0	0
	C_{34}	0	0	0	0	0	0	1	1
Σ_1		3	3	3	3	1	0	0	0
Σ_2		0	0	0	0	2	3	3	3
$\mathbf{C_i^\star, i =}$		**1**	**1**	**1**	**1**	**2**	**2**	**2**	**2**

Figure 3.5: $votingMatrix$ Computes Class Labels of Objects

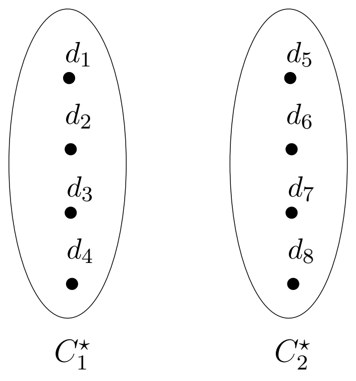


Figure 3.6: Final Clustering, $\pi^\star(\mathcal{D})$

3.5 DISCUSSION OF CLICOM

3.5.1 Advantages of CLICOM

Efficiency and Scalability

CLICOM works well with large data sets. Experimental results with data sets up to 40.000 objects are shown. Although CLICOM operates on objectwise co-associations, it does not require a $\mathcal{D} \times \mathcal{D}$ similarity matrix. CLICOM takes advantage of bitset operations to find similarity between a pair of clusters.

```
**Input**: $B$: Cluster Blocks
**Output**: $\pi^\star(\mathcal{D})$: Final Clustering
Initialize $|B| \times |\mathcal{D}|$ size $VotingMatrix$ filled with $0$s ;
Initialize $\pi^\star(\mathcal{D}) = \emptyset$ ;
for $m \leftarrow 1$ to $|B|$ do
    Initialize $\pi^\star_m = \emptyset$ ;
foreach cluster block $B^\star_k \in B$ do
    foreach cluster $C_i \in B^\star_k$ do
        foreach object $d_j \in C_i$ do
            $VotingMatrix_{kj} \leftarrow VotingMatrix_{kj} + 1$ ;
for $j \leftarrow 1$ to $|\mathcal{D}|$ do
    $m = max_k(VotingMatrix_{kj})$, $k = \{1, \ldots, |B|\}$ ;
    Add object $d_j$ to $\pi^\star_m$ ;
for $m \leftarrow 1$ to $|B|$ do
    Add $\pi^\star_m$ to $\pi^\star(\mathcal{D})$ ;
return $\pi^\star(\mathcal{D})$
```

Algorithm 8: $majorityVoter$ Places Objects into Final Clusters

Improved accuracy

Strong nature of cliques and a co-association based similarity measure improve accuracy of the consensus function.

Varying Numbers of Clusters and Clusterings

A collection of clusterings, $\Pi(\mathcal{D})$, is the main input to CLICOM. Our algorithm works well with arbitrary number of clusterings and clusters.

Automatically Computed Number of Final Clusters

Most of the clustering and combining multiple clustering algorithms require the number of final clusters in advance. However, CLICOM automatically computes the number of clusters in the final clustering with respect to its input parameters. Number of cliques and their properties, such as ranking and overlapping, affect the number of clusters in the final output clustering.

Termination Condition

The algorithm terminates automatically when all the objects are assigned to a cluster.

3.5.2 Shortcomings of CLICOM

The principal shortcoming of CLICOM is inherited from clique finding algorithms. As mentioned earlier, conventional clique finding algorithms perform poorly on large or dense graphs. Since CLICOM operates on graphs representing cluster similarities, generally these graphs tend to be small and sparse. But, in rare cases we may encounter large or dense graphs. To address this problem, we propose a novel, output-sensitive clique finding algorithm in the next section which can find satisfactory number cliques in a short amount of time.

3.6 OUTPUT-SENSITIVE CLIQUE FINDING ALGORITHM

An efficient algorithm which finds all the cliques in a graph has exponential time complexity with respect to number of vertices in the graph. In Tomita et al. (2006), Ostergard (1999), several clique finding algorithms are benchmarked for varying number of vertices and densities.

For saving run-time, clique definition may be relaxed to obtain *almost clique*s or the number of cliques may be limited. However, our proposal is a novel algorithm which obeys to clique definition, but saves time by limiting the number of cliques to $|V|^2$. Still, our aim is to accomplish a good quality final clustering.

Theorem 1. *Let A be an adjacency matrix which represents a graph $G = (V, E)$. A* ***clique matrix*** *$\mathcal{K}$ is defined as follows:*

I-) $\mathcal{K}$ is a $|V| \times |V|$ size, symmetric matrix.

II-) Initial state of $\mathcal{K}$ is represented by $\mathcal{K}^0$. Middle states are $\mathcal{K}^1, \mathcal{K}^2, ..., \mathcal{K}^{|V|-1}$, the final state is represented by $\mathcal{K}^{|V|}$.

III-) Sequential transitions occur from $\mathcal{K}^0$ to $\mathcal{K}^{|V|}$.

IV-) $\mathcal{K}^0_{ij} = \{v_i, v_j\}$ *if* $A_{ij} = 1$. $\mathcal{K}^0_{ij} = \emptyset$ *if* $A_{ij} = 0$.

V-) $\mathcal{K}^0$ *to* $\mathcal{K}^{|V|}$ *are computed using Algorithm 9.*

VI-) Each entry of $\mathcal{K}^{|V|}$*, which is not empty set, is a clique.*

VII-) There can be at most $\frac{|V| \times |V-1|}{2}$ *cliques in* $\mathcal{K}^{|V|}$.

We propose an output-sensitive method in Algorithm 9 for finding $\mathcal{K}$ in an efficient way. Number of cliques are bounded by the size of clique matrix, $\mathcal{K}$, which implies output-sensitivity.

```
Input: $A$: $|V| \times |V|$ Adjacency Matrix
Output: Set of Cliques
// Build $\mathcal{K}^0$
foreach $A_{ij} \in A$ do
    if $A_{ij} = 1$ then
        $\mathcal{K}^0_{ij} = \{v_i, v_j\}$ ;
    else
        $\mathcal{K}^0_{ij} = \emptyset$ ;
// create neighbors set
for $i \leftarrow 1$ to $|V|$ do
    $Neighbors_i = \emptyset$ ;
    for $j \leftarrow 1$ to $|V|$ do
        if $A_{ij} = 1$ and $i \neq j$ then
            Add $v_j$ to $Neighbors_i$ ;
// Sequentially Construct $\mathcal{K}^1 \ldots \mathcal{K}^{|V|}$
foreach pivot vertex $v_p$, $p = 1, 2, ..., |V|$ do
    // Initialize $\mathcal{K}^p$ with $\emptyset$ entries
    foreach $\mathcal{K}^{p-1}_{ij} != \emptyset$ do
        if $\mathcal{K}^{p-1}_{ij} \subseteq Neighbors_p$ then
            $\mathcal{K}^p_{ij} = \mathcal{K}^{p-1}_{ij} \cup \{v_p\}$ ;
        else
            $\mathcal{K}^p_{ij} = \mathcal{K}^{p-1}_{ij}$ ;
// Extract Cliques
$Cliques = \emptyset$ ;
for $i \leftarrow 1$ to $|V|$ do
    for $j \leftarrow i + 1$ to $|V|$ do
        if $\mathcal{K}^{|V|}_{ij} \neq \emptyset$ then
            Add $\mathcal{K}^{|V|}_{ij}$ to $Cliques$ ;
return $Cliques$
```

Algorithm 9: FastCliquer Algorithm

Theorem 3.6.1. *Let $\mathcal{K}$ be the clique matrix of a graph $G = (V, E)$. Entries of $\mathcal{K}^{|V|}$ which are not $\emptyset$ are cliques of G.*

Proof. We have to show that the entries of $\mathcal{K}^{|V|}$ are complete: there exists an edge between each vertex pair, and maximal: the set of vertices cannot be extended with another vertex.
Base Case: Entries of $\mathcal{K}^0$ are complete. This is immediate from the definition of $\mathcal{K}^0$.
Inductive Step: Let us assume entries of $\mathcal{K}^n$ are complete, for some n, $0 < n < |V|$. We show that the entries of $\mathcal{K}^{n+1}$ are also complete. All the neighbors of a vertex v_p can be represented by a set $Neighbors_p$. If $\mathcal{K}^n_{ij} \subseteq Neighbors_p$ holds (line 13), there is an edge between v_p and every vertex in $\mathcal{K}^n_{ij}$. Therefore $\mathcal{K}^n_{ij}$ can be extended by v_p, i.e. $\mathcal{K}^{n+1}_{ij} = \mathcal{K}^n_{ij} \cup v_p$ (line 14). If $|Neighbors_p \cap \mathcal{K}^n_{ij}| = |\mathcal{K}^n_{ij}|$ does not hold, extension does not take place $\mathcal{K}^{n+1}_{ij} = \mathcal{K}^n_{ij}$ (line 16). In either case $\mathcal{K}^{n+1}_{ij}$ is complete, thus entries of $\mathcal{K}^{n+1}$ are complete.
Conclusion: We know that the entries of $\mathcal{K}^{|V|}$ are complete. Following, we show that entries of $\mathcal{K}^{|V|}$ are maximal. An entry $\mathcal{K}^{|V|}_{ij}$ can not be extended with a new vertex v_p, since line 11 considered every possible $v \in V$. Thus, $\mathcal{K}^{|V|}_{ij}$ is maximal. As conclusion, we know that entries of $\mathcal{K}^{|V|}$ are maximal, and complete. Therefore, entries of $\mathcal{K}^{|V|}$ which are not $\emptyset$ are cliques of G. $\square$

Theorem 3.6.2. *FastCliquer Algorithm is output-sensitive.*

Proof. Let $\mathcal{K}$ be the clique matrix of a graph $G = (V, E)$. From Theorem 3.6.1, entries of $\mathcal{K}^{|V|}$ which are not $\emptyset$ are cliques of G. Since $\mathcal{K}^{|V|}$ is a symmetric matrix, there can be at most $\frac{|V|^2-|V|}{2}$ cliques. Therefore, FastCliquer Algorithm is output-sensitive. $\square$

Theorem 3.6.3. *Time complexity of FastCliquer Algorithm is $O(|V|^3)$.*

Proof. Let $\mathcal{K}$ be the clique matrix of a graph $G = (V, E)$. $\mathcal{K}^0$ has $(|V|^2 - |V|)/2$ entries. In order to obtain $\mathcal{K}^{|V|}$, all the entries of $\mathcal{K}^0$ are possibly updated by all $v \in V$, which makes $(|V|^3 - |V|^2)/2$ computations. $\square$

Theorem 3.6.4. *Space complexity of FastCliquer Algorithm is $O(|V|^2)$.*

Proof. Let $\mathcal{K}$ be the clique matrix of a graph $G = (V, E)$. $\mathcal{K}^p$ has $(|V|^2 - |V|)/2$ entries, $0 \leq p < |V|$. $\mathcal{K}^{p+1}$ can be written on $\mathcal{K}^p$, since we only need final state of the clique matrix $\mathcal{K}^{|V|}$. $\square$

Algorithm 9 performs naive pivot selection; more sophisticated methods for pivot selection, which may yield larger or better cliques, can be investigated. Extension as shown in line 14 is performed with a vertex v_p, if v_p's neighbors are already in an entry of clique matrix. Note that this satisfies completeness. This extension is performed with every possible (pivot) vertex, which satisfies maximality. Therefore, each entry in $\mathcal{K}^{|V|}$ matrix is a clique by being maximally complete.

FastCliquer in Algorithm 9 is an eligible substitute for findCliques method in Algorithm 6. Extensive experimental studies show that by using FastCliquer instead of Bron-Kerbosch implementation, considerable speed gain is achieved, while maintaining good quality final clusterings.

A	C_{11}	C_{12}	C_{13}	C_{21}	C_{22}	C_{31}	C_{32}	C_{33}	C_{34}
C_{11}	0	0	0	1	0	1	1	0	0
C_{12}	0	0	0	1	0	0	1	1	0
C_{13}	0	0	0	0	1	0	0	1	1
C_{21}	1	1	0	0	0	1	1	0	0
C_{22}	0	0	1	0	0	0	0	1	1
C_{31}	1	0	0	1	0	0	1	0	0
C_{32}	1	1	0	1	0	1	0	0	0
C_{33}	0	1	1	0	1	0	0	0	1
C_{34}	0	0	1	0	1	0	0	1	0

(a)

K	C_{11}	C_{12}	C_{13}	C_{21}	C_{22}	C_{31}	C_{32}	C_{33}	C_{34}
C_{11}	X	$\emptyset$	$\emptyset$	P	$\emptyset$	P	P	$\emptyset$	$\emptyset$
C_{12}		X	$\emptyset$	R	$\emptyset$	$\emptyset$	R	S	$\emptyset$
C_{13}			X	$\emptyset$	T	$\emptyset$	$\emptyset$	T	T
C_{21}				X	$\emptyset$	P	P	$\emptyset$	$\emptyset$
C_{22}					X	$\emptyset$	$\emptyset$	T	T
C_{31}						X	P	$\emptyset$	$\emptyset$
C_{32}							X	$\emptyset$	$\emptyset$
C_{33}								X	T
C_{34}									X

(b)

Figure 3.7: (a) Adjacency Matrix, A, and (b) $\mathcal{K}^{|V|}$ matrix for the graph in Figure 3.3(b). In K, $P = \{C_{11}, C_{21}, C_{31}, C_{32}\}$, $T = \{C_{13}, C_{22}, C_{33}, C_{34}\}$, $R = \{C_{12}, C_{21}, C_{32}\}$, $S = \{C_{12}, C_{33}\}$

Example 1. *FastCliquer is demonstrated in Figure 3.7, which shows the adjacency and clique matrices for the cluster-wise similarity graph in Figure 3.3(b).*

All cliques $\{C_{11}, C_{21}, C_{31}, C_{32}\}$, $\{C_{13}, C_{22}, C_{33}, C_{34}\}$, $\{C_{12}, C_{21}, C_{32}\}$, $\{C_{12}, C_{33}\}$ are successfully reported.

3.7 EXPERIMENTAL EVALUATIONS

In this section, we present experimental results of CLICOM in comparison to some other related methods. We begin with explaining the clustering validity measures and the data sets we have used in our experiments.

3.7.1 Measuring Clustering Validity

Goodness of final clusterings produced by CLICOM can be evaluated by cluster validity measures such as Adjusted Rand Index (ARI) Hubert and Arabie (1985), and Normalized Mutual Information (NMI) Strehl and Ghosh (2002). These clustering validity measures are explained below.

Adjusted Rand Index (ARI)

We use ARI for comparing quality of clusterings obtained by CLICOM with the real class labels. Given a final clustering $\pi^\star(\mathcal{D}) = \{C_1^\star, C_2^\star, \ldots, C_{|\pi^\star(\mathcal{D})|}^\star\}$ and the original set of classes, $\pi^o(\mathcal{D}) = \{C_1^o, C_2^o, \ldots, C_{|\pi^o(\mathcal{D})|}^o\}$, where $C_i^\star \cap C_j^\star = \emptyset$ for $1 \leq i, j \leq |\pi^\star(\mathcal{D})|$, and $C_i^o \cap C_j^o = \emptyset$ for $1 \leq i, j \leq |\pi^o(\mathcal{D})|$ with variables in Figure 3.8 referring to;

$$p = |\pi^\star(\mathcal{D})|, r = |\pi^o(\mathcal{D})|$$

$$n_{ij} = |C_i^o \cap C_j^\star|$$

$$n_{i.} = \sum_{j=1}^{p} n_{ij}$$

$$n_{.j} = \sum_{i=1}^{r} n_{ij}$$

ARI is formulated as follows:

$$\frac{\sum_{i,j} \binom{n_{ij}}{2} - \left(\sum_i \binom{n_{i.}}{2} \sum_j \binom{n_{.j}}{2}\right) / \binom{n}{2}}{\frac{1}{2}\left(\sum_i \binom{n_{i.}}{2} + \sum_j \binom{n_{.j}}{2}\right) - \left(\sum_i \binom{n_{i.}}{2} \sum_j \binom{n_{.j}}{2}\right) / \binom{n}{2}}$$

Figure 3.8: Abbreviations for ARI

Class $\setminus$ Cluster	$C_1^\star$	$C_2^\star$	$\dots$	$C_p^\star$	**Sums**
C_1^o	n_{11}	n_{12}	$\dots$	n_{1p}	$n_{1.}$
C_2^o	n_{21}	n_{22}	$\dots$	n_{2p}	$n_{2.}$
$\vdots$	$\vdots$	$\vdots$		$\vdots$	
C_r^o	n_{r1}	n_{r2}	$\dots$	n_{rp}	$n_{r.}$
Sums	$n_{.1}$	$n_{.2}$		$n_{.p}$	$n_{..} = n$

ARI takes its maximum value at 1, which indicates perfect match between two clusterings $\pi^\star(\mathcal{D})$ and $\pi^o(\mathcal{D})$.

Normalized Mutual Information (NMI)

We use as well an information-theoretic measure, NMI Strehl and Ghosh (2002), in a supervised manner, by comparing a final clustering $\pi^\star(\mathcal{D})$ with $\pi^o(\mathcal{D})$ having original class membership information. This measure computes mutual information between the two clusterings and normalize it with their total entropies.

$$NMI(\pi^\star(\mathcal{D}), \pi^o(\mathcal{D})) = \frac{MI(\pi^o(\mathcal{D}), \pi^\star(\mathcal{D}))}{\sqrt{H(\pi^o(\mathcal{D}))H(\pi^\star(\mathcal{D}))}}$$

$$= \frac{\sum_{i=1}^{|\pi^\star(\mathcal{D})|} \sum_{j=1}^{|\pi^o(\mathcal{D})|} |C_i^\star \cap C_j^o| \log\left(\frac{|\mathcal{D}||C_i^\star \cap C_j^o|}{|C_i^\star||C_j^o|}\right)}{\sqrt{\left(\sum_{i=1}^{|\pi^\star(\mathcal{D})|} |C_i^\star| \log\frac{|C_i^\star|}{|\mathcal{D}|}\right)\left(\sum_{j=1}^{|\pi^o(\mathcal{D})|} |C_j^o| \log\frac{|C_j^o|}{|\mathcal{D}|}\right)}}$$

NMI is often used in Combining Multiple Clusterings literature. Similar to ARI, NMI takes its maximum value at 1. We use NMI along with ARI in order to show that our good results are not mere coincidence.

3.7.2 Data Sets

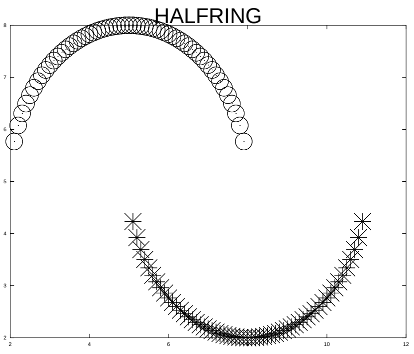
HALFRING, CURVE, SYNTH10C, SYNTH4C1, SYNTH4C2, and SYNTH4C4 are synthetic data sets. HALFRING data set contains 118 objects and two clusters. CURVE data set has 192 objects in two clusters. SYNTH10C is a 10-cluster data set obtained by using a Gaussian mixture model by using the software provided in *dbkgroup.org/handl/generators*. SYNTH4C1, SYNTH4C2, and SYNTH4C4 are large data sets each having 4-clusters with Gaussian distributions. These data sets are shown in Figure 3.9.

2D2K and 8D5K data sets are obtained from *strehl.com* and used in Strehl and Ghosh (2002) : 2D2K is synthetically generated and contains 500 points in each of its two Gaussian clusters with means (0.227, 0.077) and (0.095, 0.323) and diagonal covariance matrices with 0.1 for all diagonal elements. 8D5K contains 1000 points from five multivariate Gaussian distributions (200 points each) in 8-dimensional space. The clusters all have the same variance (0.1), but different means. Means were drawn from a uniform distribution within the unit hypercube. The data sets are illustrated in Figure 3.10.

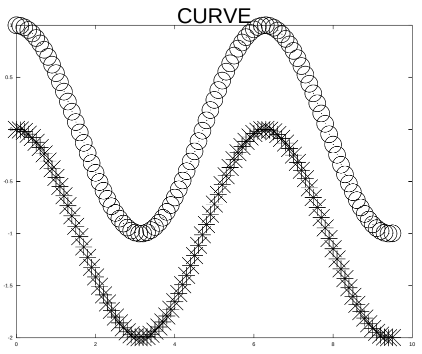
Rest of the test data sets are obtained from the UCI's machine learning repository Asuncion and Newman (2007), which are multi-dimensional with various properties. Table 3.1 shows properties of all the test data sets, where the last column shows the original number of clusters. Table 3.2 provides information about clusterings on these data sets that are used as input by CLICOM and other methods in comparison.

Table 3.1: Properties of Test Data Sets

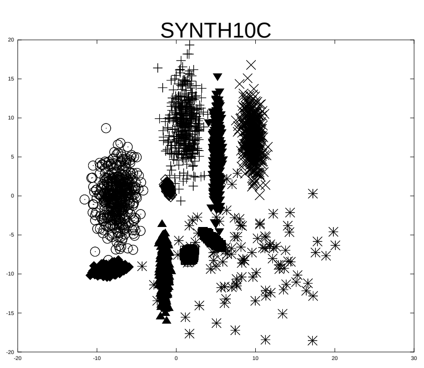
Data Set	$\lvert Objects\rvert$	$\lvert Attributes\rvert$	$\lvert Clusters\rvert$
HALFRING	118	2	2
CURVES	192	2	2
IRIS	150	4	3
GLASSIDE	214	9	6
2D2K	1000	2	2
8D5K	1000	8	5
IMAGESEG	2310	18	7
SYNTH10C	3630	2	10
SIGNFORM	5000	21	3
SYNTH4C1	10000	3	4
SYNTH4C2	20000	3	4
SYNTH4C4	40000	3	4



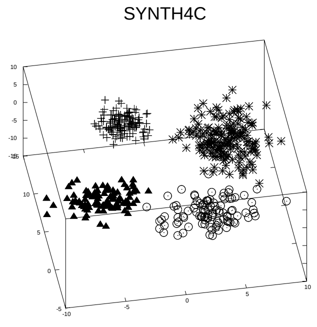
(a)



(b)

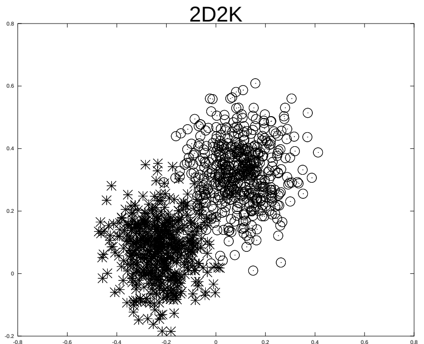


(c)

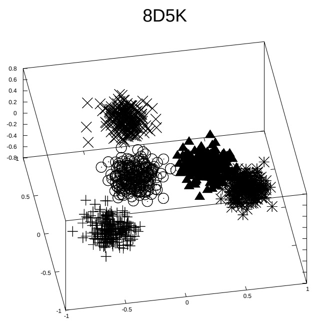


(d)

Figure 3.9: HALFRING, CURVE, SYNTH10C Data Sets. Randomly Sampled 500 Objects of SYNTH4C Family.



(a)



(b)

Figure 3.10: 2D2K and 8D5K (projected on 3 Principal Components) Data Sets

Table 3.2: Input Clustering Properties on Data Sets

			ARI	
Data set	$\lvert Clusterings\rvert$	$\lvert Clusters\rvert$	Worst	Best
HALFRING	3	2 - 5	0.41	0.81
CURVES	3	3 - 6	0.33	0.78
IRIS	5	2 - 6	0.54	0.76
GLASSIDE	5	5 - 7	0.58	0.73
2D2K	3	2	0.66	0.79
8D5K	5	5	0.55	0.74
IMAGESEG	5	7	0.44	0.45
SYNTH10C	20	5 - 14	0.56	0.78
SIGNFORM	5	3 - 5	0.44	0.47
SYNTH4C1	4	3 - 6	0.73	0.82
SYNTH4C2	9	3 - 6	0.57	0.93
SYNTH4C4	10	3 - 6	0.56	0.87

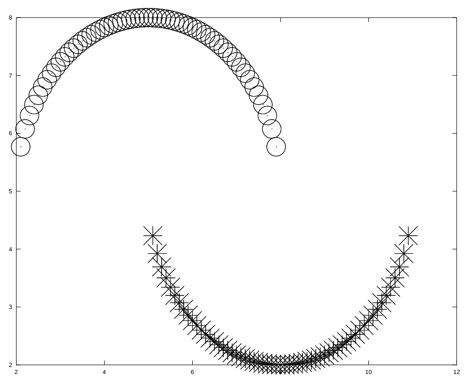
3.7.3 Experimental Results

Figure 3.11 demonstrates CLICOM on HALFRING data set. Figures 3.11(b)-3.11(d) show input clusterings and their quality. Using the input provided in Figures 3.11(b)-3.11(d), CLICOM generates a final clustering having a much better overall quality as shown in Figure 3.11(a).

On IRIS, which is a real data set, CLICOM produces a final clustering with 100% accuracy as shown in Figure 3.12(a) by using the input clusterings shown in Figures 3.12(b)-3.12(f). All the input clusterings are obtained by k-Means with k having values between 2 and 6. Although the clusterings provided by k-Means, even with the correct number of clusters (k=3), have notably low quality, CLICOM is able to combine these clusterings into a final clustering having perfect quality.

On SYNTH10C data set, which has 10 clusters, similar successful results of CLICOM are shown in Figure 3.13.

For all the data sets, Table 3.3 shows qualities of final clusterings obtained by using both Bron-Kerbosch and FastCliquer algorithms. This table demonstrates two very important points: (1) CLICOM produces better quality final clusterings than the collection of input clusterings (2) FastCliquer is as accurate as Bron-Kerbosch implementation for the purpose of combining multiple clusterings.


(a) Output of CLICOM with ARI=1.00

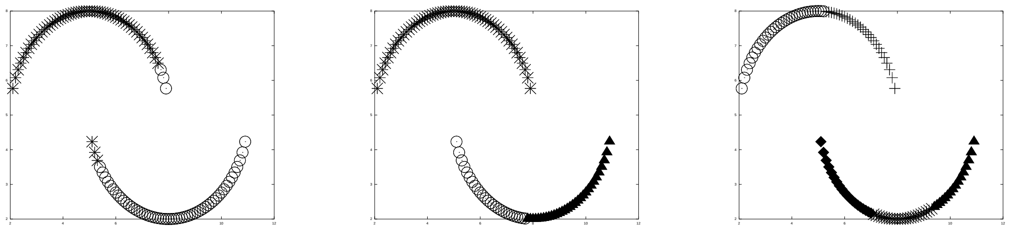
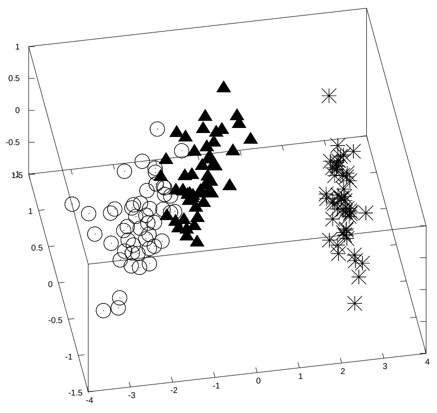

(b) $|\pi_1(\mathcal{D})|$=2 and ARI=0.81 (c) $|\pi_2(\mathcal{D})|$=3 and ARI = 0.75 (d) $|\pi_3(\mathcal{D})|$=5 and ARI=0.41

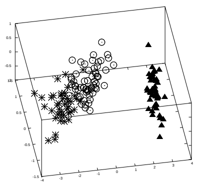
Figure 3.11: Result of CLICOM on 3 Input Clusterings on HALFRING

Main advantage of using FastCliquer instead of Bron-Kerbosch is speed, which can be observed in Tables 3.4 and 3.5(b). Considerably shorter run-times are obtained on especially data sets having large and dense similarity graphs.

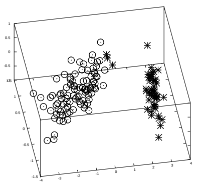
Instead of using cluster-wise similarity graphs, we also investigated the benefits and shortcomings of providing object-wise similarity graphs to CLICOM. On object-wise similarity graphs, Bron-Kerbosch algorithm is infeasible for large number of objects, as expected. FastCliquer algorithm solves this problem by producing cliques in practical run-times. Object-wise similarity graphs require a size $|\mathcal{D}| \times |\mathcal{D}|$ similarity matrix generation and storage, which is a major computational and storage requirement. Counter-intuitively, we also observed much less accurate results in our experiments when using object-wise similarity graphs. All put together, we do not suggest using CLICOM on objects. Accuracy results and run-times are displayed in Table 3.5.



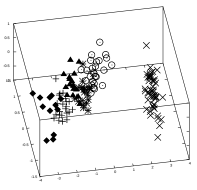
(a) Output of CLICOM with ARI=1.00



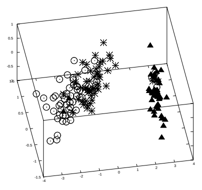
(b) $|\pi_1(\mathcal{D})|$=3 and ARI=0.75



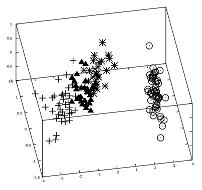
(c) $|\pi_2(\mathcal{D})|$=2 and ARI=0.54



(d) $|\pi_3(\mathcal{D})|$=6 and ARI=0.64



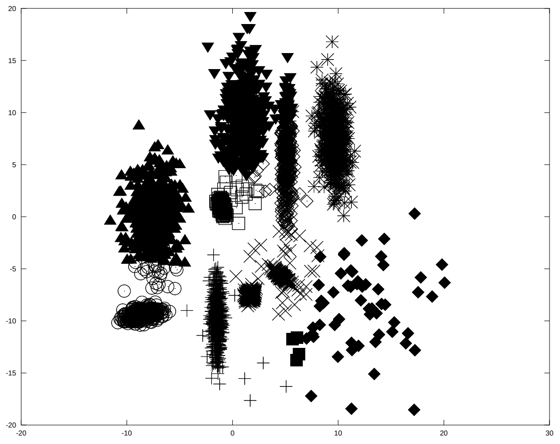
(e) $|\pi_4(\mathcal{D})|$=3 and ARI=0.74



(f) $|\pi_5(\mathcal{D})|$=4 and ARI=0.76

Figure 3.12: Result of CLICOM on 5 Input Clusterings on IRIS

Table 3.6 displays test results of the related work. EAC (with single link) and CSPA methods run out of memory on SYNTH4C2 and SYNTH4C4 data sets. Note that EAC, MCLA, CSPA and HGPA all need the number of clusters in the final clustering as input. Although providing the input correctly creates an unfair advantage to EAC, MCLA, CSPA and HGPA, we provided correct number of clusters. On most of the data sets, CLICOM generates better quality final clusterings than EAC, MCLA, CSPA and HGPA. Best results


(a) Output of CLICOM with ARI = 0.86

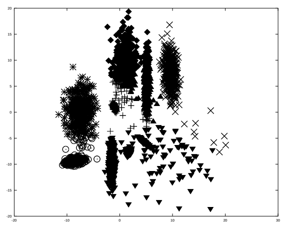
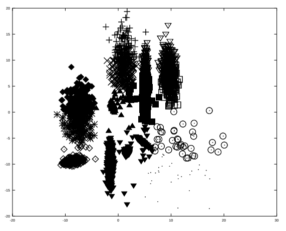
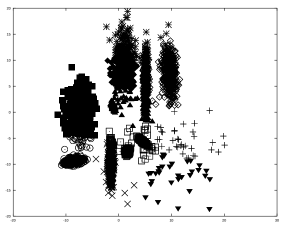



(b) $|\pi_2(\mathcal{D})|$=7 and ARI=0.61 (c) $|\pi_9(\mathcal{D})|$=12 and ARI=0.58 (d) $|\pi_{16}(\mathcal{D})|$=10 and ARI=0.7

Figure 3.13: Result of CLICOM on 20 Input Clusterings on SYNTH10C, Some Input Clusterings Shown in (b)-(d)

of these algorithms and CLICOM are shown in Figure 3.14, clearly CLICOM is superior. CLICOM obtains these good quality results very fast, which is shown in the same figure.

3.8 CONCLUSIONS AND FUTURE WORK

Providing the best sparsification value automatically to CLICOM is an open research area, since good sparsification yields good results. Best value of overlapping percentage (δ) in cliquesToClusters needs further investigation. We experimentally found that $\delta = 10\%$ produces good results on our test data sets. In FastCliquer algorithm, ordering pivots in the best possible way for producing large cliques is also a future research direction.

In this chapter, we propose a novel algorithm for combining a collection of clusterings into a final clustering having better overall quality. Our method, CLICOM, is based on solid foundations since it utilizes cliques for generating clusters, and cliques form very strong

Table 3.3: CLICOM Validity Results Using Bron-Kerbosch and FastCliquer Algorithms

		Bron-Kerbosch			FastCliquer		
Data set	θ	$\lvert\pi^\star\rvert$	ARI	NMI	$\lvert\pi^\star\rvert$	ARI	NMI
HALFRING	0.03	1	0.00	0.00	1	0.00	0.00
	0.33	2	**1.00**	*1.00*	2	**1.00**	*1.00*
	0.67	4	0.71	0.77	3	0.77	0.83
CURVE	0.17	2	**0.98**	*0.96*	2	**0.98**	*0.96*
	0.33	3	0.76	0.80	3	0.76	0.80
IRIS	0.40	2	0.57	0.76	2	0.57	0.76
	0.50	3	**1.00**	*1.00*	3	**1.00**	*1.00*
	0.60	3	0.92	0.91	3	1.00	1.00
	0.70	5	0.85	0.89	4	0.87	0.91
GLASSIDE	0.20	5	0.92	0.95	5	0.92	0.95
	0.26	6	**1.00**	*1.00*	6	**1.00**	*1.00*
	0.40	6	0.96	0.95	5	0.97	0.97
	0.60	5	0.74	0.66	2	0.52	0.68
2D2K	0.07	1	0.00	0.00	1	0.00	0.00
	0.50	2	**0.79**	*0.69*	2	**0.79**	*0.69*
8D5K	0.20	4	0.76	0.86	4	0.76	0.86
	0.40	5	**0.96**	*0.94*	5	**0.96**	*0.94*
	0.60	6	0.88	0.88	5	0.96	0.94
IMAGESEG	0.30	7	**0.89**	*0.86*	7	**0.89**	*0.86*
	0.40	7	0.80	0.79	7	0.88	0.86
SYNTH10C	0.40	9	0.79	0.87	9	0.79	0.87
	0.50	10	**0.86**	*0.90*	10	**0.86**	*0.90*
	0.60	12	0.75	0.84	11	0.68	0.81
SIGNFORM	0.26	3	0.87	0.81	3	**0.87**	*0.81*
	0.40	3	**0.89**	*0.83*	3	0.86	0.81
SYNTH4C1	0.25	4	**0.99**	*0.98*	4	**0.99**	*0.98*
SYNTH4C2	0.60	4	**0.98**	*0.97*	4	**0.98**	*0.97*
SYNTH4C4	0.75	4	**0.98**	*0.97*	4	**0.98**	*0.97*

clusters. We study several clique finding algorithms and provide a new clique finding method that is fast and accurate. CLICOM can be used on finer and coarser granularities by providing object-wise or cluster-wise similarity graphs. On cluster-wise similarity graphs, CLICOM produces remarkably good quality final clusterings due to its similarity measure which is based on object co-associations. Extensive experimental results on real and artificial data sets show that CLICOM scales very well, and produces output as fast as the other state-of-the-art methods.

Table 3.4: CLICOM Run-Time Results Using Bron-Kerbosch and FastCliquer Algorithms

		Run-time (ms)	
Data set	θ	*Bron-Kerbosch*	*FastCliquer*
HALFRING	0.33	20	**5**
CURVE	0.17	35	**14**
IRIS	0.50	52	**25**
GLASSIDE	0.26	85	**40**
2D2K	0.50	25	**8**
8D5K	0.40	110	**50**
IMAGESEG	0.30	125	**70**
SYNTH10C	0.50	2,825	**1,650**
SIGNFORM	0.40	100	**95**
SYNTH4C1	0.25	105	**45**
SYNTH4C2	0.60	350	**130**
SYNTH4C4	0.75	760	**420**

Table 3.5: On Object-wise Similarity Graph, CLICOM (a) Validity Results (b) Run-Time Results

Data set	θ	$\|\pi^\star\|$	ARI	NMI
IRIS	0.40	3	0.76	0.77
GLASSIDE	0.40	6	0.96	0.95
8D5K	0.20	5	0.95	0.93

(a)

		Run-time (ms)	
Data set	θ	*Bron-Kerbosch*	*FastCliquer*
IRIS	0.40	13,500	**225**
GLASSIDE	0.40	50,250	**340**
8D5K	0.20	$>$30min	**30,500**

(b)

Table 3.6: Cluster Validity Results of the Related Work (OOM: Out of Memory)

Data set	$\lvert\pi^\star\rvert$	ARI			
		EAC	MCLA	CSPA	HGPA
HALFRING	2	1.00	1.00	1.00	0.43
	3	0.80	0.68	0.44	0.50
CURVE	2	0.98	0.98	0.98	0.23
	3	0.74	0.75	0.43	0.45
IRIS	2	0.57	0.44	0.44	0.37
	3	0.64	0.88	0.98	0.29
	4	0.65	0.94	0.55	0.06
GLASSIDE	5	0.91	0.93	0.48	0.31
	6	0.91	0.98	0.51	0.30
2D2K	2	0.79	0.79	1.00	0.00
	3	0.76	0.76	0.45	0.60
8D5K	5	0.61	0.96	0.98	0.00
	6	0.61	0.93	0.64	0.55
IMAGESEG	6	0.00	0.69	0.64	0.23
	7	0.00	0.88	0.85	0.54
SYNTH10C	9	0.64	0.80	0.67	0.54
	10	0.64	0.76	0.60	0.52
	12	0.67	0.81	0.57	0.52
SIGNFORM	2	0.00	0.50	0.38	0.01
	3	0.00	0.87	0.88	0.09
	4	0.00	0.87	0.51	0.00
SYNTH4C1	3	0.62	0.80	0.70	0.23
	4	0.99	0.74	0.69	0.00
SYNTH4C2	3	OOM	0.80	OOM	0.20
	4	OOM	0.98	OOM	0.00
SYNTH4C4	3	OOM	0.80	OOM	0.48
	4	OOM	0.98	OOM	0.00

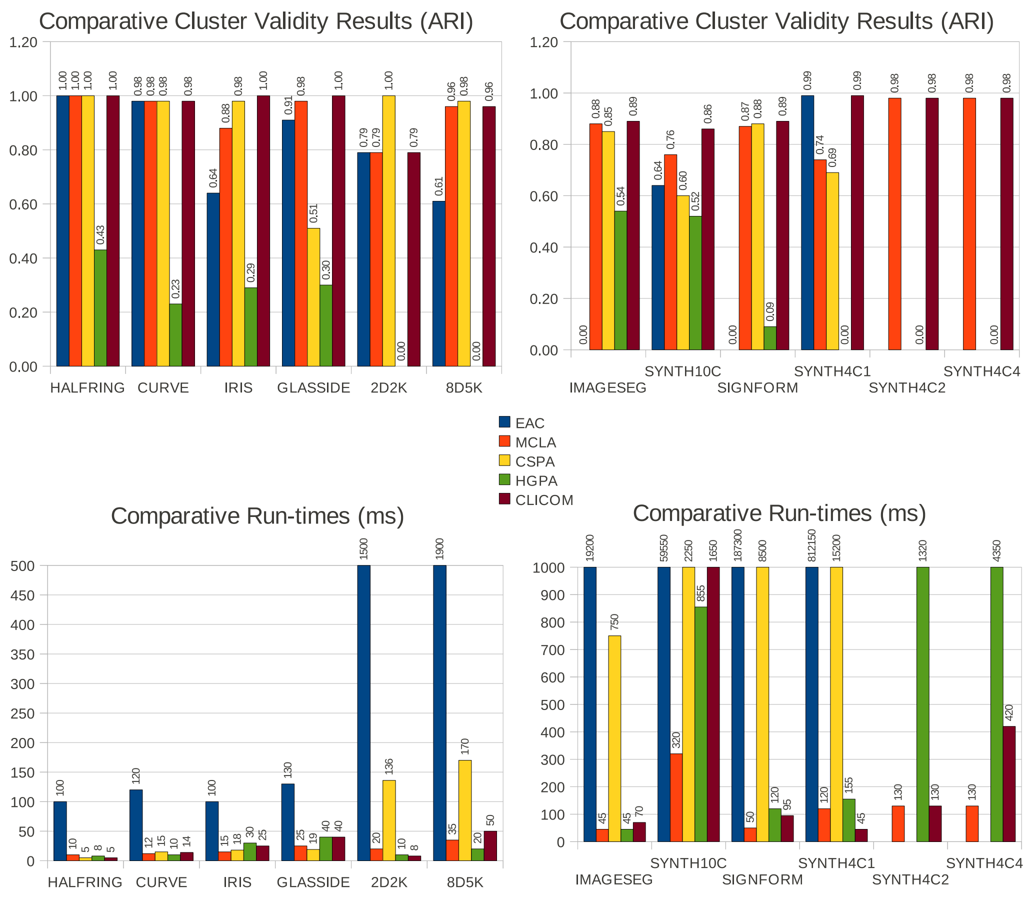


Figure 3.14: Comparison of Validity and Run-time Results

4. APPROXIMATIVE COMPUTING OF DISTANCES BY RANDOM HASHING

In this chapter, we propose a technique for approximate computation of inter-object distances in binary data sets. Our approach is based on Locality Sensitive Hashing, scales up well with the number of objects and it is especially useful for parallel computing environments. We obtain multiple weak clusterings of a data set by randomly selecting a number of projections on its attributes and then grouping objects into buckets based on the common values of these projections. For each pair of objects, occurrences in the same bucket are counted and the exact Hamming distance is approximated based on the portion of co-occurrences in the buckets. Next, we parallelize the computation using mainly two different schemes. The first assigns each subspace to a single process calculating its parts of the co-occurrence matrix and afterwards adds up the complete co-occurrence matrix over all subspaces. The second method exchanges results between each process during computation.

4.1 INTRODUCTION

Locality Sensitive Hashing (LSH), introduced in Indyk and Motwani (1998) and Andoni and Indyk (2008), can be used for an approximate calculation of distances between the tuples of a table by using randomized hash functions. A close variant of LSH which works best with the Hamming distance is described in Gionis et al. (1999).

We propose a method for approximating the distance matrix for binary data sets represented by bit vectors. The core idea is to choose m k-dimensional subspaces randomly and consider a bucket for each possible bit vector in this subspace. Then, the vectors are hashed into the matching buckets and, for each pair of objects the occurrences in the same bucket are counted. The exact Hamming distance is approximated based on the portion of co-occurrences in the m subspaces. We parallelize this computation using different schemes.

Our data set is a binary table $\mathcal{D}$, having N distinct objects and a set I that consists of n distinct attributes. A set $K \subseteq I$ of k attributes, designated as a *probe* and chosen randomly, defines a random hashing function f_K by assigning to a tuple t the numerical binary equivalent of the projection of t on the set K, $t[K]$.

Each hashing function produces a partition of the set of tuples; each block of this partition consists of tuples that collide under that hashing function.

LSH is used for clustering the Web in Haveliwala et al. (2000). In Koga et al. (2007) it is used to enhance the agglomerative hierarchical clustering of the single link method Sibson (1973b). Both of these techniques rely on the same idea provided by LSH: close objects are likely to collide under a high number of randomly chosen hashing functions. Both of these techniques compute the real distances between objects residing in the same blocks.

LSH-Link algorithm Koga et al. (2007) has $O(N^2)$ time complexity as the classical single link method and it works as follows. Database $\mathcal{D}$ is hashed into m partitions by using randomly generated functions of LSH on k attributes. In the first phase of the algorithm distances between all pairs of tuples $(\mathbf{u}, \mathbf{v})$ residing in the same blocks are computed, and all the pairs of tuples having distance at most r are merged at once. Note that this pairwise computation takes place on every block of every partition. If after the first phase there is more than one cluster, the algorithm proceeds to the second phase by selecting a new projection size k' such that $k' < k$, and a new distance value r' such that $r' > r$. LSH-Link hashes the whole database $\mathcal{D}$ again by using the new values r', k'. It merges the pair of clusters with respect to r'. If there is more than one cluster, the algorithm proceeds to the next phase by selecting new values r'', k'' and by repeating all the calculations. This continues until there is only one cluster. Authors of Koga et al. (2007) point that in a phase several clusters may be merged as opposed to merging only two clusters in classical single link algorithm. Note that if the initial distance r is not chosen carefully the LSH-Link algorithm may take many phases, therefore yielding many redundant hashing and pairwise distance computations. Furthermore, in the worst case this algorithm computes N^2 distances.

The clustering algorithms proposed in Haveliwala et al. (2000) and Koga et al. (2007) focus on finding the approximate set of near neighbors $\mathsf{ANN}(\mathbf{u})$ of an object $\mathbf{u}$, followed by finding real near neighbors of $\mathbf{u}$ by computing the actual distances $d(\mathbf{u}, \mathbf{v})$ for all $\mathbf{v} \in \mathsf{ANN}(\mathbf{u})$. Note that some of the real neighbors of $\mathbf{u}$ may be missed because LSH does not guarantee to put all the close objects in the same blocks.

We aim for a distinct goal, namely an efficient, approximative computation of the distance matrix of the set of objects using LSH, which allows us to use a variety of standard clustering algorithms.

Parallel and distributed computing techniques are able to solve big and complicated problems by using a variety of divide-and-conquer techniques. In this chapter, we introduce several parallel data mining programming methodologies that are applicable in two widely used architectures: shared disk cluster environment, and shared memory architectures Flynn (1972).

Preliminary results are presented in Mimaroglu and Simovici (2008), and the chapter is structured as follows. Section 4.2 examines the relation between randomly generated hash function collisions and distances. In Section 4.3, we present the algorithms and implementation guidelines. Experimental environments and test results are presented in Section 4.4. A final section contains our conclusions and plans for future work.

4.2 COLLISIONS AND DISTANCES

In this section, we examine the relation between randomly generated hash function collisions and distances between objects.

A *binary data collection* is a sequence $\mathcal{D} = (\mathbf{t}_1, \ldots, \mathbf{t}_N)$ of tuples, where $\mathbf{t}_i \in \{0,1\}^n$.

Let $K = \{i_1, \ldots, i_k\} \subseteq \{1, \ldots, n\}$. The *projection of a tuple* $\mathbf{t} \in \{0,1\}^n$ *on* K is the tuple $\mathbf{t}[K] = (t_{i_1}, \ldots, t_{i_k})$. The K*-projection of the binary data collection* $\mathcal{D}$ is the binary data collection $\mathcal{D}[K] = (\mathbf{t}_1[K], \ldots, \mathbf{t}_N[K])$.

Each K-projection of $\mathcal{D}$ generates a function $f_K : \{1, \ldots, N\} \longrightarrow \mathbb{N}$, where $f_K(r)$ is the binary equivalent of the sequence $\mathbf{t}_r[K]$. This can be seen in Figure 4.1, where the function $f_{\{i_1,i_3,i_5\}}$ for the binary data collection shown in Part (a) is given in Part (b) of the figure. $f_{\{i_1,i_3,i_5\}}$ creates a partition with 8 blocks; 2 of these are empty as shown in Figure 4.2.

The Hamming distance between two tuples $\mathbf{u}, \mathbf{v} \in \{0,1\}^n$ is given by

$$d(\mathbf{u}, \mathbf{v}) = |\{i \in \{1, \ldots, n\} \mid u_i \neq v_i\}|,$$

where $\mathbf{u} = (u_1, \ldots, u_n)$ and $\mathbf{v} = (v_1, \ldots, v_n)$.

$\mathcal{D}$

r	i_1	i_2	i_3	i_4	i_5
1	1	0	0	1	1
2	0	1	1	0	0
3	1	0	1	0	0
4	1	1	0	1	0
5	0	1	1	1	1
6	0	0	1	1	1
7	1	0	1	0	1
8	1	1	0	0	1
9	0	1	1	1	0

(a)

$\mathcal{D}[K]$

r	$f_K(r)$
1	5
2	2
3	6
4	4
5	3
6	3
7	7
8	5
9	2

(b)

Figure 4.1: A binary collection and the hashing function f_K for $K = \{i_1, i_3, i_5\}$.

000 {}	*001* {}	*010* {2,9}	*011* {5,6}
100 {4}	*101* {1,8}	*110* {3}	*111* {7}

Figure 4.2: All the blocks created by f_K for $K = \{i_1, i_3, i_5\}$. Block descriptors, and corresponding row numbers are shown.

Suppose that the set of attributes K that defines a probe is chosen at random. There are $\binom{n}{k}$ such choices if $|K| = k$. A collision takes place between two rows **u** and **v** if the chosen k attributes are among the $n - d$ attributes on which **u** and **v** are equal, where $d = d(\mathbf{u}, \mathbf{v})$ is the Hamming distance between **u** and **v**. There are $\binom{n-d}{k}$ such choices for the set I. Therefore, for any two rows $\mathbf{u}, \mathbf{v}$ of $\mathcal{D}$ the collision probability for f_K, that is, the probability that $f_K(\mathbf{u}) = f_K(\mathbf{v})$ is

$$p = \frac{\binom{n-d}{k}}{\binom{n}{k}}.$$

If m sets K having k elements are chosen at random, then $C(\mathbf{u}, \mathbf{v})$ the total number of collisions that occur in this experiment is a binomially distributed variable with the distribution $B(m,p)$. Thus, the expected number of collisions is

$$E(C(\mathbf{u},\mathbf{v})) = m\frac{\binom{n-d}{k}}{\binom{n}{k}}$$

if $k \leq n-d$, which is typically the case. If $k > n-d$ a collision is impossible and $p = 0$. It is clear that the smaller the distance $d(\mathbf{u},\mathbf{v})$, the larger the number of collisions will be.

Using Stirling's Formula we can write

$$\begin{aligned}
\frac{\binom{n-d}{k}}{\binom{n}{k}} &= \frac{\frac{(n-d)!}{k!(n-d-k)!}}{\frac{n!}{k!(n-k)!}} \\
&= \frac{(n-d)!}{n!}\frac{(n-k)!}{(n-d-k)!} \\
&\approx \frac{(n-d)^{n-d+0.5}(n-k)^{(n-k+0.5)}}{n^{(n+0.5)}(n-d-k)^{n-d-k+0.5}} \\
&= \left(\frac{n^2-nd-nk+dk}{n^2-nd-nk}\right)^{n+0.5} \cdot \\
&\quad \left(\frac{n-d-k}{n-d}\right)^{d} \cdot \left(1-\frac{d}{n-k}\right)^{k}.
\end{aligned}$$

For moderately large values of n the first two factors are close to 1. Thus, the expected value of the number of collisions is

$$E(C(\mathbf{u},\mathbf{v})) \approx m \cdot \left(1-\frac{d}{n-k}\right)^{k}$$

Let $c(\mathbf{u},\mathbf{v}) = E(C(\mathbf{u},\mathbf{v}))/m$ be the *relative number of collisions*. Then, we estimate the distance between **u** and **v** as

$$d(\mathbf{u},\mathbf{v}) \approx (n-k)(1-c(\mathbf{u},\mathbf{v})^{\frac{1}{k}}) \tag{4.1}$$

Assume that m probes with k attributes are applied, where $1 \leq m$ and $1 \leq k \leq n$. Since we deal with binary data, each attribute may take a value of either 0 or 1. Therefore, the partition that corresponds to a k-probe may contain up to 2^k blocks.

Let $n_1, \ldots, n_{2^k}$ be the sizes of the blocks that correspond to a k-probe. For each block of the partition we need to update the number of collisions of pairs. Therefore, for a block of size n_i we need to perform $\binom{n_i}{2}$ updates of the pair counters. For example, for a block having three elements $\{a, b, c\}$, the collision counts of the pairs: $(a, b), (a, c), (b, c)$ are increased by one. The average size of a block is $\frac{n}{2^k}$, so the average total time required is

$$\begin{aligned} \sum_{i=1}^{2^k} \binom{n_i}{2} &= \frac{1}{2}\left(\sum_{i=1}^{2^k} {n_i}^2 - n_i\right) \\ &= \frac{1}{2}\left(2^k\left(\frac{n}{2^k}\right)^2 - 2^k\left(\frac{n}{2^k}\right)\right) = \frac{n^2}{2^{k+1}} - \frac{n}{2}. \end{aligned}$$

The process has to be repeated for each of m probes and this requires an average time proportional to $m\left(\frac{n^2}{2^{k+1}} - \frac{n}{2}\right)$.

4.3 ALGORITHMS AND IMPLEMENTATION GUIDELINES

A clustering, which corresponds to a probe, is represented by e.g. a Java or a C++ class that is parameterized by the projection size. Components of a clustering include its clusters and members of these clusters.

To produce a clustering, k attributes are randomly selected and the objects are projected on the selected set of attributes. A cluster C consists of those objects that have the same projection $\mathbf{p} \in \{0, 1\}^k$ on the set of attributes that constitutes the probe. The value of bit vector $\mathbf{p}$ is the descriptor of the cluster. The cluster itself is represented by a bit vector $\mathbf{b}_C \in \{0, 1\}^N$, where $(\mathbf{b}_C)_i = 1$ if and only if the object $\mathbf{u}_i$ belongs to the cluster C.

Clusters (blocks) do not overlap. The identifiers of the objects are placed into appropriate clusters according to the descriptors.

The number of clusterings m is determined by the user and is passed as an argument to the implementation. Both the number of clusterings (which equals the number of probes m) and the width k of the probes are set to positive integers by the user. Note that even for small values of k, the probability of selecting the same probe twice is rather small because there are $\binom{n}{k}$ probes and m is typically much smaller than $\binom{n}{k}$.

All clusterings are populated in one scan of the database as follows. Each clustering may have at most 2^k non-empty clusters. First, empty clusterings are initialized, then each object in the database $\mathcal{D}$ is passed to all the clusterings. Each clustering projects the object on its own randomly selected attributes and then places the object in the appropriate cluster according to the cluster descriptors.

Assume a clustering projects on first, fifth, and tenth attributes, then the object $\mathbf{1}001\mathbf{0}1101\mathbf{0}$, is placed in cluster 4 of this clustering. Similarly, if another clustering projects on fourth, sixth, and seventh attributes, then the same object $100\mathbf{1}0\mathbf{11}010$, is placed in cluster 7 of this clustering. This computation takes place for each data object. In one scan of database $\mathcal{D}$, m clusterings, each having 2^k clusters can be generated efficiently.

We use an $N \times N$ matrix referred to as the *simultaneous occurrence matrix* to keep track of the number of collisions of each of the object pairs. After obtaining the clusterings, each cluster in every clustering is scanned once and the occurrence matrix component that corresponds to each pair $(\mathbf{u}, \mathbf{v})$ of objects in $\mathcal{D}$ that co-occur in the same cluster is incremented by 1. Note that there are at most $m2^k$ clusters to scan.

For example, assume that we have a 5-object database $\mathcal{D}$ with $n = 4$ attributes, the width of the probes is $k = 1$, and we have $m = 3$ clusterings, whose bit vectors are

$$\begin{pmatrix} 0 \\ 1 \\ 1 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 0 \end{pmatrix}, \text{ and } \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}.$$

The first bit vector indicates that the two buckets that correspond to the probe consist of the objects $\{u_1, u_5\}$ and $\{u_2, u_3, u_4\}$, respectively; the other bit vectors are constructed in a similar way. The simultaneous occurrence matrix is shown below. For example, objects 2 and 4 occur in the same clusters 3 times.

	1	**2**	**3**	**4**	**5**
1	3	0	2	0	2
2	0	3	1	3	1
3	2	1	3	1	1
4	0	3	1	3	1
5	2	1	1	1	3

Using the simultaneous occurrence matrix, the approximate distance matrix can be computed using Formula (4.1). For the same example, the distance matrix is shown below.

	1	**2**	**3**	**4**	**5**
1	0	3	1	3	1
2	3	0	2	0	2
3	1	2	0	2	2
4	3	0	2	0	2
5	1	2	2	2	0

In a distributed parallel computing environment (in our case, a Beowulf cluster), each worker node reads the database file from the shared disk and creates a collection of bit vectors representing the projected columns. Objects having the same values on the projected columns are placed in the same clusters. At each node, simultaneous occurrence matrix is filled with 0s and 1s (representing collision). To reduce the message size between processors we use the upper right half of SOM, which is referred as Simultaneous Occurrence Vector (SOV). Note that the number of simultaneous occurrences cannot be greater than number of nodes in the cluster. Kambadur et al. (2006) and Tansey and Tilevich (2008) offer good solutions for reducing the total message size. In order to keep the SOV size reasonably low we use type **char** (8 bits). There are 124 nodes in our cluster, therefore in the resulting SOV the largest value may be 124 and this number can be represented by 8 bits.

Algorithm 10 is for computing SOVs in the cluster environment. This algorithm is graphically represented in Figure 4.5.

SOVs obtained from each worker node are summed into a resultant SOV. Instead of summing all the SOVs sequentially in the master node, we use the built-in MPI function *MPI_Reduce*. MPI_Reduce is handled in parallel, and in logarithmic time, therefore it is

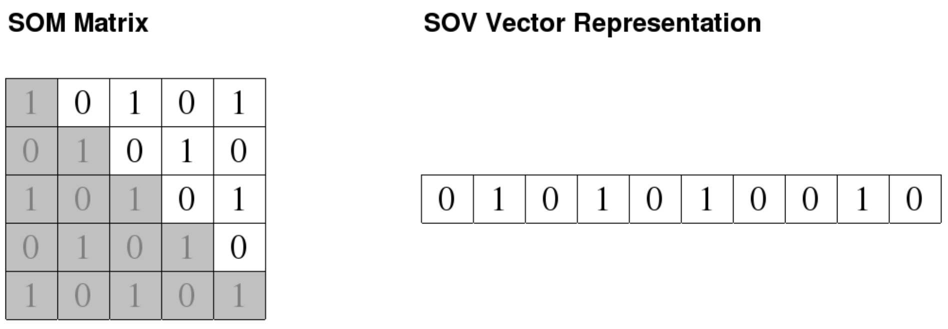


Figure 4.3: In a Cluster Environment Use SOV to Reduce MPI Message Size

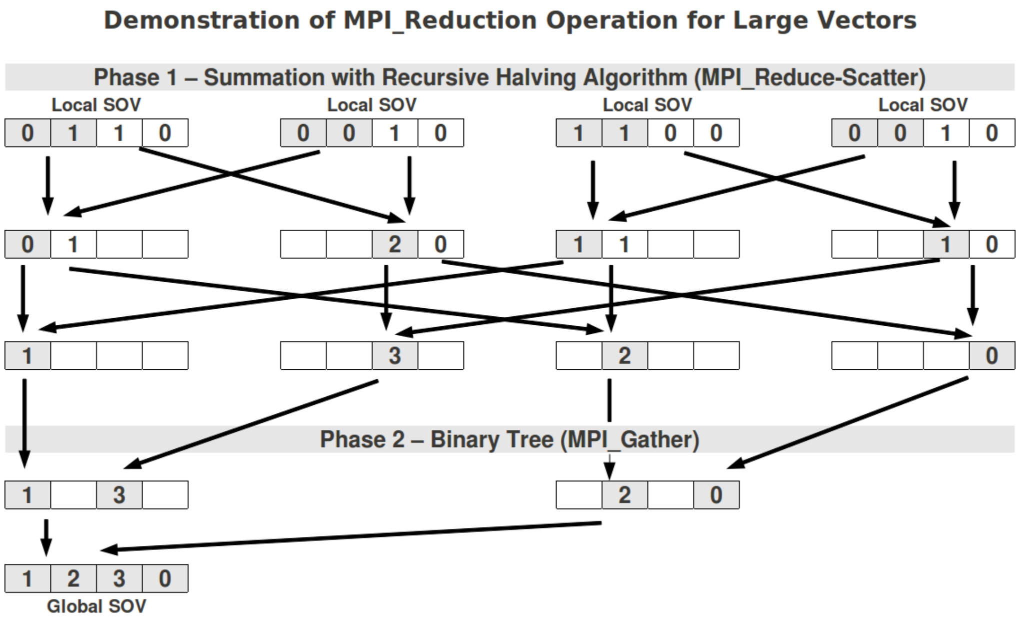

Demonstration of MPI_Reduction Operation for Large Vectors
Phase 1 – Summation with Recursive Halving Algorithm (MPI_Reduce-Scatter)
Local SOV
Local SOV
Local SOV
Local SOV
Phase 2 – Binary Tree (MPI_Gather)
Global SOV

Figure 4.4: MPI_Reduce Adding 4 SOVs in Parallel

very advantageous. Full details of the MPI_Reduce depends on the message size and the MPI implementation. Using Thakur et al. (2005) and Dhillon and Modha (2000) a conceptual cost model for MPI collective operations is developed and shown in Figure 4.4. An alternative idea is to compress each SOV, and decompress the SOVs at receiving nodes, then perform summation. For achieving this, along with compression functions MPI_Pack, MPI_Unpack routines are used. SOVs gets much smaller when they are compressed. However, there is a large overhead to compress each SOV at the sender node, and to decompress each SOV at the receiver side. Note that the alternative approach reduces the total message size in the cluster noticeably, but it incurs overhead for constantly compressing and decompressing. Experimental results showed us that using MPI_Reduce with original SOVs yields better performance.

```
Input: $\mathcal{D}$: database, $k$: projection size, $m$: number of projections
Output: $SOV$: Simultaneous Occurrence Vector
// There are $m$ worker nodes
foreach worker node do
    Represent $\mathcal{D}$ vertically by a collection of bit vectors in the main memory, refer to this
    bit vector collection as $Dbv$ ;
    On $Dbv$, create a random projection of size $k$ by choosing $k$ columns randomly;
    Form a clustering $\mathcal{C}$ containing $2^k$ clusters ;
    // Some of the clusters in $\mathcal{C}$ may be empty
    Place each tuple in $Dbv$ into corresponding cluster according to the values in
    randomly chosen columns;
    Initialize a Simultaneous Occurrence Vector SOV;
    foreach cluster $K \in \mathcal{C}$ do
        foreach pair $(i, j) \in K$ do
            Set value of (i,j) in SOV to 1;
Sum all SOVs in parallel using, MPI_Reduce ;
```
Algorithm 10: Parallel Algorithm to Compute Simultaneous Occurrence Vector

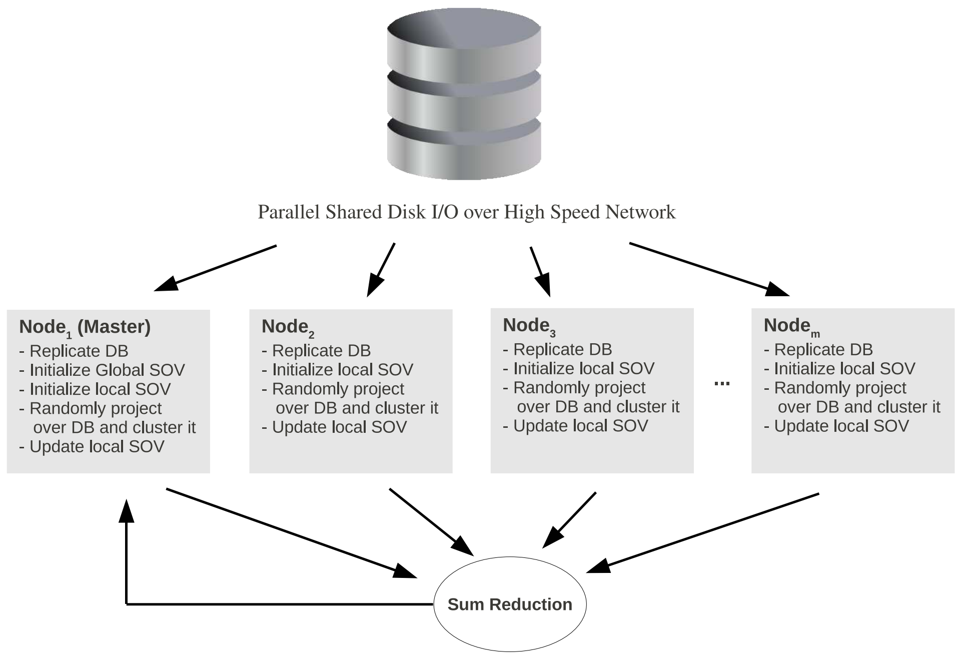


Figure 4.5: Computing SOV in Cluster Environment

As the final procedure for computing the Approximate Distance Matrix, Formula (4.1) is applied to the resultant SOV as shown in Algorithm 11: Instead of performing this operation sequentially on one node, we divide SOV into m equal fragments and apply

Formula (4.1) on m different worker nodes for the corresponding fragments. Finally, the master node converts it to the Approximate Distance Matrix (ADM) or uses it as is. Conversion just maps the elements of SOV into ADM, therefore it is computationally cheap. For distributing SOV fragments into m worker nodes we use MPI_Scatter; for collecting the fragments the MPI_Gather functionality is used in the cluster environment.

```
Input: $SOV$: Simultaneous Occurrence Vector, $m$: number of nodes
Output: $ADM$: Approximate Distance Matrix
// Fragment SOV into $m$ contiguous vectors of size $n$ each
n = size(SOV) / m;
Scatter each fragment $SOV_f$, $f \in \{1, \ldots, m\}$ to m worker nodes;
// Apply distance approximation formula
foreach $SOV_f$, $f \in \{1, \ldots, m\}$ do
    foreach element e in $SOV_f$ do
        if e != 0 then
            e = applyDistanceFormula(e) ;
Gather $SOV_f$, $f \in \{1, \ldots, m\}$ in the master node to form $SOV$;
foreach element e in master $SOV$ do
    Map e in ADM;
```

Algorithm 11: Parallel Algorithm for Computing Approximate Distance Matrix (ADM)

For very large databases we noticed that the total message size in Algorithm 10 becomes problematic. Therefore, we created an alternative method as shown in Algorithm 12. In this algorithm, each node operates on a $\frac{|SOV|}{m}$ size message, where m is the total number of nodes. The algorithm passes each SOV fragment $m-1$ times in a circular motion between the worker nodes. For achieving this we created a virtual circular topology of m worker nodes as shown in Figure 4.6. In this setting, each node decides on the columns to project randomly, and creates a fragment of SOV. For example $Node_1$ has the first fragment containing first $\frac{|SOV|}{m}$ entries, and $Node_m$ has the last $\frac{|SOV|}{m}$ entries. After projecting, each node updates the SOV message it has, then passes this SOV message to its right, and receives a new SOV message from its left. The circular message passing between nodes is completed in $m-1$ iterations. This design makes sure that anytime in the system maximum message size is $|SOV|$, which makes it suitable for very large databases.

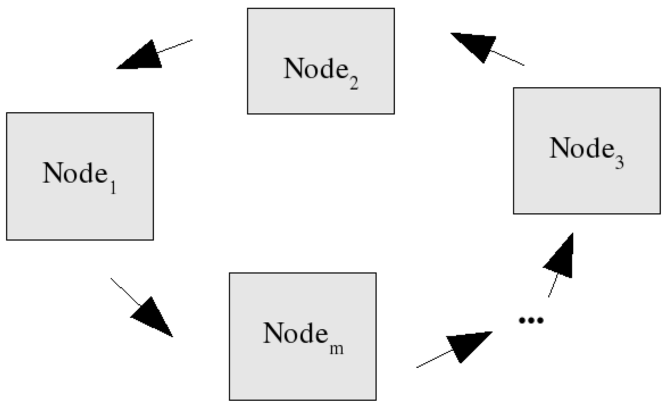


Figure 4.6: Alternative Approach for Computing SOVs in Circular Topology

```
Input: $\mathcal{D}$: database, $k$: projection size, $m$: number of projections
Output: $SOV$: Simultaneous Occurrence Vector
// There are $m$ worker nodes
foreach worker node do
    Represent $\mathcal{D}$ vertically by a collection of bit vectors in the main memory, refer to this
    bit vector collection as $Dbv$ ;
    On $Dbv$, create a random projection of size $k$ by choosing $k$ columns randomly;
    Form a clustering $\mathcal{C}$ containing $2^k$ clusters ;
    // Some of the clusters in $\mathcal{C}$ may be empty
    Place each tuple in $Dbv$ into corresponding cluster according to the values in
    randomly chosen columns;
    Initialize a Simultaneous Occurrence Vector SOV fragment;
    Update the SOV fragment at hand ;
for 1 to m-1 do
    foreach worker node do
        Send SOV fragment to the right node ;
        Update SOV fragment received from left node ;
Combine all the SOV fragments into a global SOV ;
```

Algorithm 12: Parallel Algorithm to Compute Simultaneous Occurrence Vector with Circular Topology

4.4 EXPERIMENTAL RESULTS

Our primary testing environment is a Beowulf cluster having 124 nodes, with infiniband connectivity. Each node has a 64-bit processor with 4GB to 8GB of main memory. The cluster is equipped with parallel file system, message passing interface (MPI) Gropp et al. (1998) Tu et al. (2009), Linux operating system, and 62 1.0-GHz dual core AMD Opteron processors.

Our choice of programming language on the cluster is C++. For conducting experiments we used MPICH2 Gropp (2002), along with the gcc version 4.2. compiler, and Boost library Karlsson (2005). In the cluster, whose high level topology is given in Figure 4.7, work load balancing is performed manually.

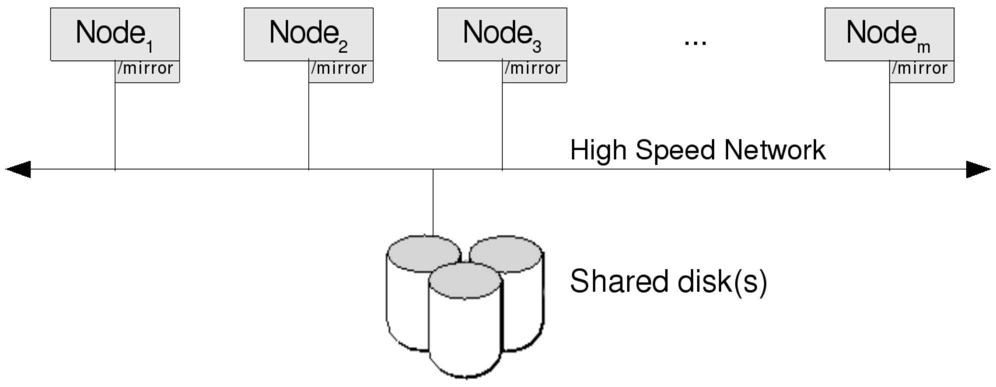


Figure 4.7: Topology of the Beowulf Cluster Having 124 Nodes

Since we are interested in implementing our algorithms on platforms with a relatively small number of processors (which are widely available) we used as a secondary experimental environment an Apple - Mac Pro having 2 Intel 3.0-GHz Xeon quad-core 64-bit processors. This server has 8 cores and 16 GB of total main memory. On this shared memory system, we use Java threads, and these threads are converted to operating system native threads by the compiler. We relied on the operating system (Mac OS X Leopard) to distribute the work evenly.

Implementation for the Beowulf cluster required more lower level work compared with the secondary testing environment. In the Beowulf cluster, we relied mostly on MPI libraries for achieving reliability, and synchronization. In the shared memory environment we used locks, and atomic operations (where possible) to solve similar problems.

The test data sets we used are randomly generated using independent uniform distributions for each bit and consisted of bit vectors of length 20 unless otherwise indicated. The average density of the 1s is 50% in each vector.

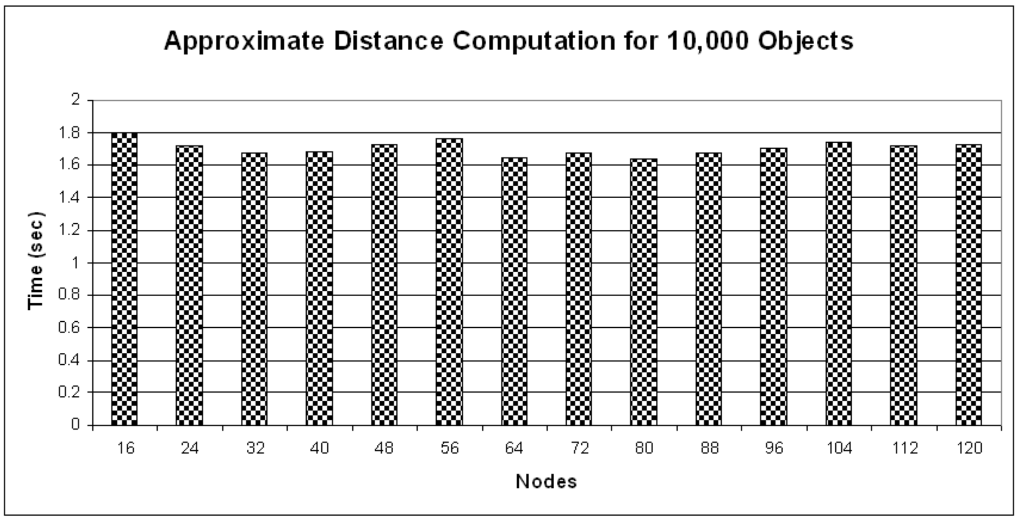


Figure 4.8: Runtime on 10,000 Objects Using Algorithms 10 and 11 in the Cluster Environment

Figure 4.8 presents the total execution time for creating the approximate distance matrix (ADM) on a database having 10,000 objects. Note that in this test and in all the remaining tests presented in this section, increasing the number of nodes produces more accurate ADMs. Results in Figure 4.8 indicate that our implementation runs fully in parallel. Running the algorithm on 96 nodes and above achieves excellent accuracy relative to the actual Hamming distance. Also, note that we are physically limited by two factors: available nodes, and the bandwidth. In Algorithm 10, each node computes an SOV, and then these SOVs are merged into a final SOV in the master node. The size of each SOV is $\frac{n(n-1)}{2}$ where n is the number of objects in the database. For example, in a database with $10,000$ objects, there are around 50 million entries in the SOV (each entry is 1 byte long for saving space). Each node sends the computed SOV through the high speed infiniband network as a message. Projection size does not effect the execution time, but the messages are reflected as overhead.

Computing the ADMs of databases having $40,000$ objects or more by using the Algorithms 10 and 11 is problematic due to the total message size. Figure 4.9 shows that for $40,000$ objects only 32 nodes could be utilized: It was impossible to use more nodes because of the network bottleneck and memory requirements. Clearly, for very large databases we need to use the alternative approach presented in Algorithm 12. Although this approach is more time consuming, it is less sensitive to total message size and mem-

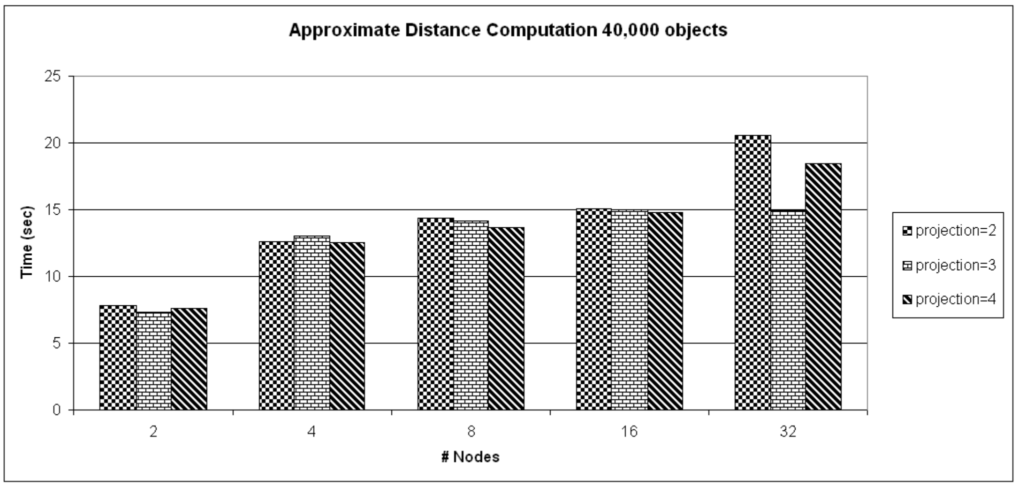


Figure 4.9: Runtime on 40,000 Objects Using Algorithms 10 and 11 in the Cluster Environment

ory requirements by keeping the network traffic at a constant level as shown in Figure 4.10.

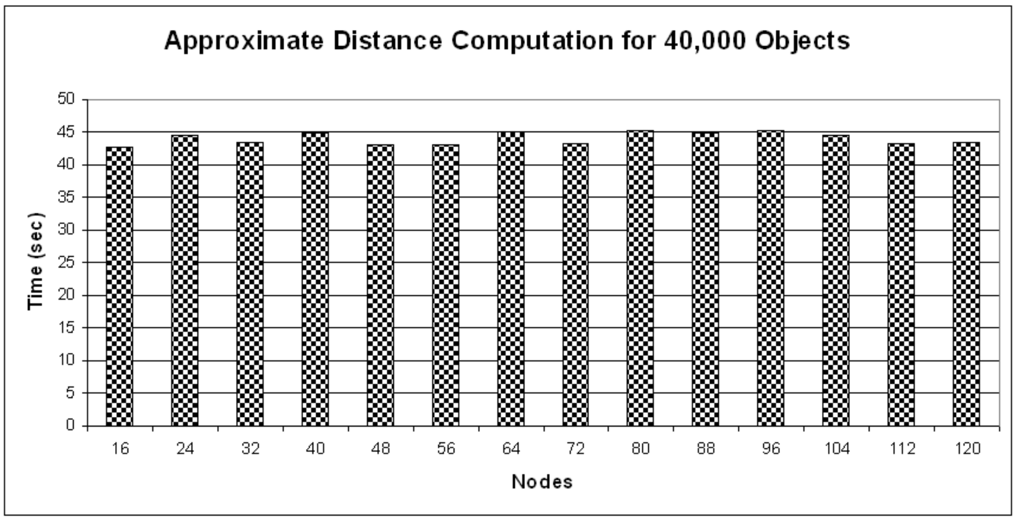


Figure 4.10: Runtime on 40,000 Objects Using Algorithms 12 and 11 (Circular Topology) in the Cluster Environment

To evaluate the accuracy of our approximation we used the cophenetic correlation coefficient Sokal and Rohlf (1962). This coefficient takes a value between 0 and 1, where a higher value implies better correlation. We calculated the cophenetic correlation coefficient between our approximate distance matrix ADM , and the Hamming distance matrix HM. The averages of the matrices ADM and HM are denoted by $\overline{d}$ and $\overline{hm}$, respectively. The coefficient is given by

$$c = \frac{\sum(ADM_{ij} - \overline{d})(HM_{ij} - \overline{hm})}{\sqrt{\sum(ADM_{ij} - \overline{d})^2 \sum(HM_{ij} - \overline{hm})^2}}.$$

In Figure 4.11, we show the cophenetic correlation coefficient for varying k and m. Note that the best experimental results are achieved when $k = 2$. Higher values of m produces better correlations and the coefficient approaches to 1 for reasonable values of m. Figure 4.12 shows that e.g. for 20, 100, and 200 probes, the cophenetic correlation coefficient is 0.773, 0.934, and 0.972 respectively. 200 probes may seem extreme, but each probe scans only 2 attributes out of the total 20 attributes. Therefore, each probe scans 10 percent of the database, and 200 probes correspond to a total of 20 full scans of the database. On the other hand, to compute a distance matrix of $10,000$ points, around $5,000$ full scans of the database are required.

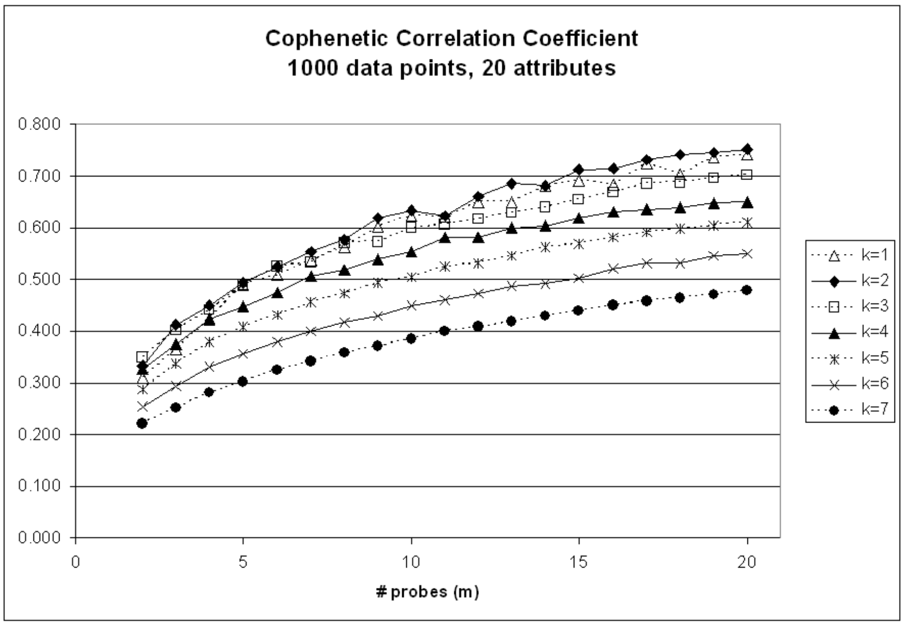


Figure 4.11: Cophenetic Correlation Coefficient on 1,000 Objects Having 20 Attributes with Varying k

On our secondary platform (an Apple - Mac Pro) which has only 8 nodes each clustering is implemented as a Java thread which is converted to an operating system native thread by the compiler. On a database having 15,000 objects, we computed approximate distance matrices for $k = 4$ and varying number of nodes (probes). In Figure 4.13 we report the total execution time. The results are as expected: total time stays almost stable when increasing number of nodes.

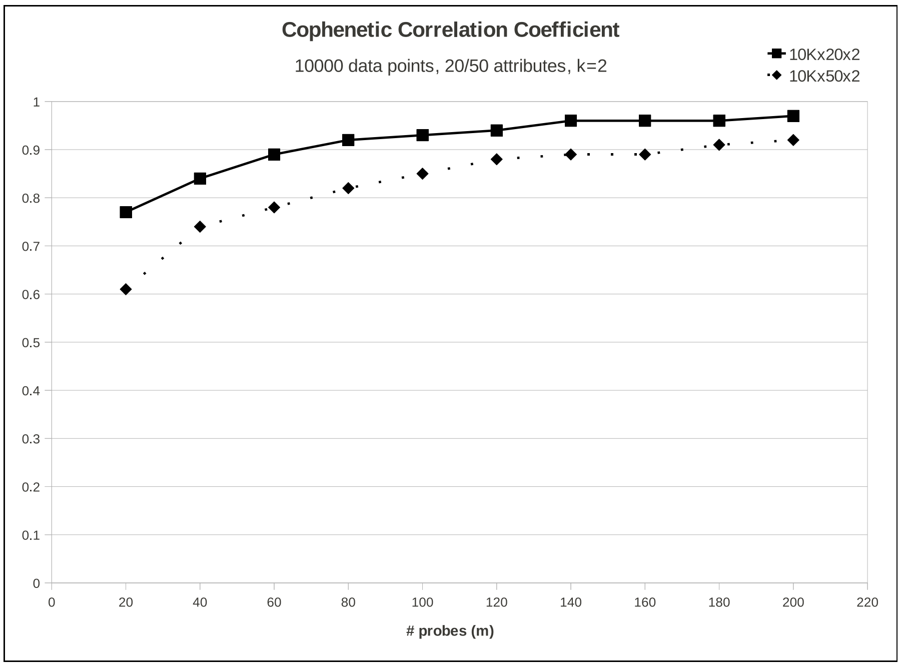


Figure 4.12: Cophenetic Correlation Coefficient on 10,000 Objects Having 20 and 50 Attributes with $k = 2$ and Varying Values of m (nodes)

Nodes	Total Time (sec)
2	2.2
3	2.1
4	2.1
5	2.3
6	2.3
7	2.4
8	2.5

Figure 4.13: Runtime on 15,000 Objects in the Secondary Testing Environment

Figure 4.14 shows run-time results of approximate distance computation in comparison to sequential computation of Hamming distance. We also implemented a parallel computation method for Hamming distance. In both implementations, we use bit sets and take the cardinality of XOR operation on bit sets, which is the fastest way to compute Hamming distance. In the parallel algorithm, only upper half of the Hamming distance matrix is computed because the matrix is symmetric. When using m nodes, the matrix is divided into m equal parts and distributed to m nodes for computing. Table 4.1 presents time complexities of algorithms comparatively.

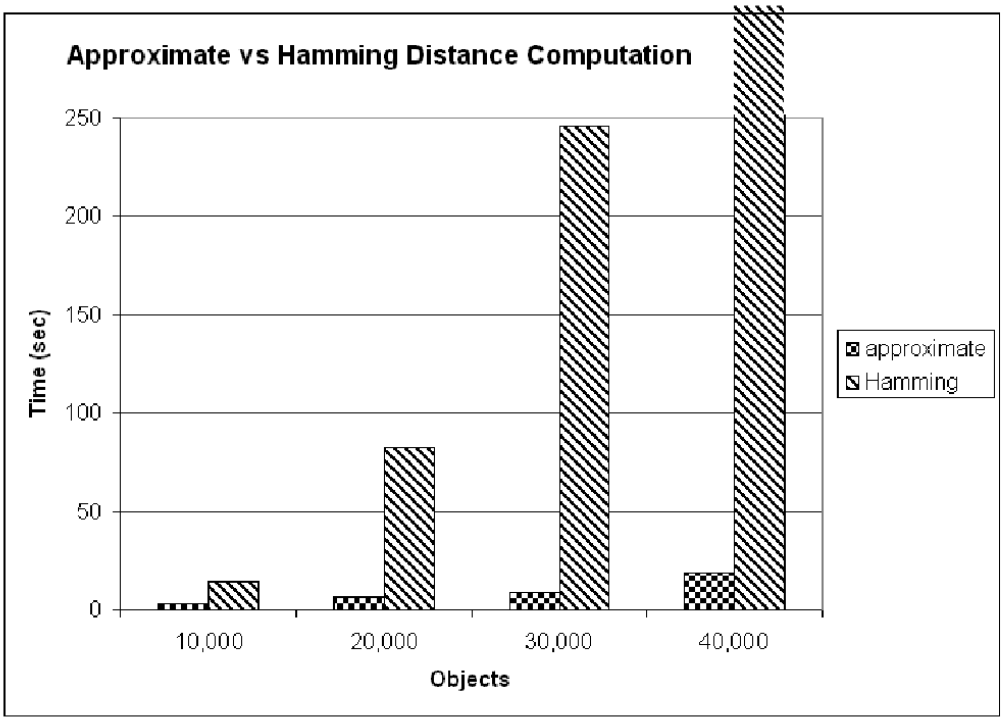


Figure 4.14: Comparing Run-time Results

Table 4.1: Comparison of Algorithm Complexities

Algorithm	**Complexity**	**Explanations**
Sequential Hamming Distance	$O(\|\mathcal{D}\|^2 \times \|I\|)$	$\|I\|$: cardinality of set of attributes
Sequential Approximate Distance	$m(\frac{\|\mathcal{D}\|^2}{2^{k+1}} - \frac{\|\mathcal{D}\|}{2})$	See Section 4.2
Parallel Approximate Distance (Algo. 10 + 11)	$\frac{\|\mathcal{D}\|^2}{2^{k+1}} - \frac{\|\mathcal{D}\|}{2} + \alpha \times \log \|\mathcal{D}\|^2 + \gamma$	with added complexity of MPI collective operations
Parallel Hamming Distance	$O(\frac{\|\mathcal{D}\|^2 \times \|I\|}{m}) + \beta \times \log \|\mathcal{D}\|^2 + \lambda$	m is $O(\|\mathcal{D}\|)$ and complexity of MPI collective operations is added

4.5 CONCLUSIONS AND FUTURE WORK

Computing the distance matrix of a data set is a fundamental problem in clustering. We present an efficient, approximative approach for distance calculations in dense binary data sets that relies on randomized hash functions known as Locality Sensitive Hashing (LSH). Implementation guidelines, and several methods which are suitable for distributed and shared memory architectures are discussed. Experimental results demonstrate that our parallel methods are comparably fast and accurate. We contemplate developing new parallel clustering ensemble algorithms that will combine several clusterings into a single superior clustering in an efficient manner.

5. CONCLUSIONS

Clustering is a major research field in Data Mining, Machine Learning, and Pattern Recognition. An important research trend in clustering is Combining Multiple Clusterings, which seeks methods of consensus among available clusterings of a data set with the goal to obtain new better final clustering. A detailed literature survey for Combining Multiple Clusterings is presented in Chapter 1. This thesis introduces three novel methods which contribute to clustering research, more specifically to the area of Combining Multiple Clusterings. These methods are presented in Chapters 2, 3, and 4. Our methods involve efficient and scalable computation techniques, also with low memory requirements which enable their application to large data sets.

5.1 FASTFIT

FastFit is a novel binary method for fast computation of intra-cluster and inter-cluster similarities of a clustering, $\pi^{\star}(\mathcal{D})$. It is based on pairwise object co-associations in a set of Multiple Clusterings, $\Pi(\mathcal{D})$. Nonetheless, it does not require a $|\mathcal{D}| \times |\mathcal{D}|$ co-association matrix for this computation. Furthermore, by representing each cluster using a bit vector, we utilize fast cluster-wise operations and low memory requirements of binary operations.

FastFit can be used as an objective function for optimization-based consensus solutions like evolutionary algorithms. As a cluster validity measure, FastFit is used to evaluate cluster cohesion and cluster separation.

Test results with data sets up to 1M data objects demonstrate the effectiveness of FastFit. The method is superior to the conventional technique, in that, it is fast and efficient, and it scales to large data sets and uses incomparably less memory. FastFit is reliable since it provides a measure of cluster cohesion and separation based on object co-associations.

As a future work, we will investigate designing a novel evolutionary method for combining multiple clusterings of a large data set using FastFit.

5.2 CLICOM

CLICOM is a graph-based consensus solution for combining multiple clusterings. It finds maximally complete subgraphs, also known as cliques, in a graph representation of available input clusterings. This is a weighted graph where each vertex represents a cluster, and an edge represents the similarity between a pair of clusters. The basic idea is, since cliques constitute strong clusters, a good selection of cliques may yield a good final clustering. CLICOM finds number of clusters in the final clustering automatically with respect to a threshold value, θ. Similarity of a pair of clusters is evaluated by using a similarity measure, $ECS_{\Pi(\mathcal{D})}$, based on object co-associations. Computation of this similarity is fast and does not require a $|\mathcal{D}| \times |\mathcal{D}|$ co-association matrix. $ECS_{\Pi(\mathcal{D})}$ is shown to produce more accurate results compared to syntactical measures like Jaccard similarity.

Finding all cliques in large dense graphs is computationally overwhelming. Although, CLICOM works with clusters and a threshold, θ, is used to sparsify the similarity graph, we propose a novel output-sensitive clique finding algorithm, FastCliquer, which enables working on larger graphs more efficiently. FastCliquer is an eligible substitute for our purposes, e.g. compared to the widely used Bron-Kerbosch algorithm for finding cliques. Extensive experimental studies show that FastCliquer achieves considerable speed gain, while it still finds all or substantial amount of useful cliques.

Experimental evaluations on real and synthetic data sets are provided with benchmark results of major algorithms for combining multiple clusterings.

As in many graph-based clustering problems, automatically providing the best sparsification threshold to CLICOM is open to research, since better sparsification yields better results. Nonetheless, the algorithm is still very practical due to the exploratory nature of clustering. We plan to further investigate the proposed procedure for choosing the best cliques among others. In FastCliquer algorithm, ordering pivots in the best possible way for producing large cliques is also a future research direction.

5.3 APPROXIMATIVE COMPUTING OF DISTANCES BY RANDOM HASHING

Based on an approximation technique by Mimaroglu and Simovici (2008), we propose methods for parallel computation of inter-object distances especially in dense binary data sets. Our approach is based on Locality Sensitive Hashing, scales up well with the number of objects and is much faster than ”brute-force” computation of these distances using Hamming distance.

A data set, $\mathcal{D}$, has N distinct objects and a set I with n distinct attributes. A set $K \subseteq I$ of k attributes, designated as a probe and chosen randomly, defines a random hashing function f_K by assigning to a tuple t, the numerical binary equivalent of the projection of t on the set K. Each hashing function produces a clustering of $\mathcal{D}$. Each cluster in this clustering consists of tuples that collide under the hashing function. The relation between randomly generated hash function collisions and distances between objects is explained.

The approximation technique is especially suitable for parallel computation. We take advantage of several contemporary multiprocessor architectures and show results of different parallel algorithms on distributed- and shared-memory architectures. Experimental results show that the proposed method is fast and accurate. The work presented in this chapter provides guidelines for further work on combining weak clusterings.

5.4 OPEN RESEARCH AREAS

With an emerging need for processing huge and ever-increasing amounts of diverse data coming from e.g. the internet, audio, video and image sensors, and scientific experiments, Combining Multiple Clusterings is an important research trend in Data Clustering Jain (2010). We briefly discuss in Table 5.1, some open areas in Combining Multiple Clusterings research.

Table 5.1: Some Open Research Areas

Problem	**Description**
Determining number of clusterings, $\vert\Pi(\mathcal{D})\vert$	Number of clusterings needed to obtain a good final clustering vary with the quality of clusterings. Topchy, Law, Jain and Fred (2004) show that consensus functions based on stochastic partition generation, relabeling, voting and, median partition approaches, converge to a true underlying clustering solution as the number of partitions in the Multiple Clusterings increases.
Determining $\vert\pi^{\star}(\mathcal{D})\vert$	Automatic detection of number of clusters in the Final Clustering is a desired property, although most consensus functions take this number as an input parameter.
Combining Weak Clusterings	A weak clustering algorithm produces a clustering, which is only slightly better than a random one. Such algorithms are usually very simple and computationally inexpensive. Combining a sufficient number of such clusterings yield good clustering results. Topchy et al. (2005), Mimaroglu and Simovici (2008) provide promising results in this direction.
Working with large and very large data sets	Most algorithms show their results on very small data sets. Combining Multiple Clusterings of large data sets (High dimensional data with $\vert\mathcal{D}\vert \geq 10^4$ data objects) is not well studied.
Cluster Validation	Good and diverse unsupervised validity measures to evaluate validity of the Final Clustering is required.

REFERENCES

Books

Abraham, A., Grosan, C. and Ramos, V. (eds): 2006, *Swarm Intelligence in Data Mining*, Vol. 34 of *Studies in Computational Intelligence*, Springer.

Alpaydin, E.: 2010, *Introduction to Machine Learning*, MIT Press, Cambridge, MA.

Chapelle, O., Scholkopf, B. and Zien, A. (eds): 2006, *Semi-Supervised Learning (Adaptive Computation and Machine Learning)*, The MIT Press.

Gonzalez, R. C. and Woods, R. E.: 2006, *Digital Image Processing (3rd Edition)*, Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

Gropp, W., Huss-Lederman, S., Lumsdaine, A., Lusk, E., Nitzberg, B., Saphir, W. and Snir, M.: 1998, Mpi—the complete reference: Volume 2, the mpi-2 extensions.

Han, J. and Kamber, M.: 2005, *Data Mining: Concepts and Techniques*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Jain, A. K. and Dubes, R. C.: 1988, *Algorithms for clustering data*, Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

Karlsson, B.: 2005, *Beyond the C++ standard library*, Addison-Wesley Professional.

Kaufman, L. and Rousseeuw, P. J. (eds): 2005, *Finding Groups in Data: An Introduction to Cluster Analysis*, Wiley.

Kohonen, T., Schroeder, M. R. and Huang, T. S. (eds): 2001, *Self-Organizing Maps*, Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Lee, K. H.: 2004, *First Course On Fuzzy Theory And Applications.*, SpringerVerlag.

Michalewicz, Z.: 1996, *Genetic algorithms + data structures = evolution programs (3rd ed.)*, Springer-Verlag, London, UK.

Tan, P.-N., Steinbach, M. and Kumar, V.: 2005, *Introduction to Data Mining, (First Edition)*, Addison-Wesley Longman Publishing, Boston, MA, USA.

Periodicals

Agrawal, R., Gehrke, J., Gunopulos, D. and Raghavan, P.: 1998, Automatic subspace clustering of high dimensional data for data mining applications, *ACM-SIGMOD International Conference on Management of Data*, ACM Press, pp. 94–105.

Akkoyunlu, E.: 1973, The enumeration of maximal cliques of large graphs, *SIAM Journal on Computing* **2**(1), 1–6.

Andoni, A. and Indyk, P.: 2008, Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions, *Communications of the ACM* **51**(1), 117–122.

Asur, S., Ucar, D. and Parthasarathy, S.: 2007, An ensemble framework for clustering protein protein interaction networks, *Bioinformatics* **23**(13), i29–40.

Avogadri, R. and Valentini, G.: 2009, Fuzzy ensemble clustering based on random projections for dna microarray data analysis, *Journal of Artificial Intelligence in Medicine* **45**(2-3), 173–183.

Azimi, J., Cull, P. and Fern, X.: 2009, Clustering Ensembles Using Ants Algorithm, *Proceedings of the 3rd International Work-Conference on The Interplay Between Natural and Artificial Computation: Part I: Methods and Models in Artificial and Natural Computation. A Homage to Professor Mira's Scientific Legacy*, Springer, p. 304.

Barthelemy, J.-P. and Leclerc, B.: 1995, The median procedure for partition, *in* I. J. Cox, P. Hansen and B. Julesz (eds), *Partitioning Data Sets*, Vol. 19 of *AMS DIMACS Series in Discrete Mathematics*, AMS, pp. 3–34.

Bellman, R. E.: 2003, *Dynamic Programming (Republished)*, Dover Publications, Incorporated.

Breiman, L.: 1996, Bagging predictors, *Machine Learning* **24**(2), 123–140.

Bron, C. and Kerbosch, J.: 1973, Algorithm 457: finding all cliques of an undirected graph, *Commun. ACM* **16**(9), 575–577.

Chang-ming, Z., Guo-chang, G., Hai-bo, L., Jing, S. and Hualong, Y.: 2008, Segmentation of ultrasound image based on cluster ensemble, pp. 418 –421.

Chang, Y., Lee, D.-J., Hong, Y. and Archibald, J.: 2008, Unsupervised video shot segmentation using global color and texture information, *ISVC '08: Proceedings of the 4th International Symposium on Advances in Visual Computing*, Springer-Verlag, Berlin, Heidelberg, pp. 460–467.

Cristofor, D. and Simovici, D.: 2002, Finding median partitions using information-theoretical-based genetic algorithms, *Journal of Universal Computer Science* **8**, 153–172.

Dempster, A. P., Laird, N. M. and Rubin, D. B.: 1977, Maximum likelihood from incomplete data via the em algorithm, *Journal of the Royal Statistical Society, Series B* **39**(1), 1–38.

Dhillon, I. and Modha, D. S.: 2000, A data-clustering algorithm on distributed memory multiprocessors, *Lecture Notes in Computer Science* **1759**, 245–260.

Dietterich, T. G.: 2000, Ensemble methods in machine learning, *MCS '00: Proceedings of the First International Workshop on Multiple Classifier Systems*, Springer-Verlag, London, UK, pp. 1–15.

Duarteo, J., Jorge, F., Duarte, F., Lourenco, A. and Fred, A.: 2010, On consensus clustering validation, *S+SSPR International Workshops on Structural and Syntactic Pattern Recognition (SSPR 2010) and Statistical Techniques in Pattern Recognition (SPR 2010)*.

Elhadary, R., Tolba, A., Elsharkawy, M. and Karam, O.: 2007, An efficient and robust combined clustering technique for mining in large spatial databases, pp. 439 –445.

Ester, M., Kriegel, H.-P., Jörg, S. and Xu, X.: 1996, A density-based algorithm for discovering clusters in large spatial databases with noise, *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96)*, AAAI Press, pp. 226–231.

Faceli, K., de Souto, M. C. P., de Araujo, D. S. A. and de Carvalho, A. C. P. L. F.: 2009, Multi-objective clustering ensemble for gene expression data analysis, *Neurocomput.* **72**(13-15), 2763–2774.

Fern, X. and Brodley, C.: 2003, Random projection for high dimensional data clustering: A cluster ensemble approach, *Proc. 20th International Conference on Machine Learning (ICML'03)*, Washington.

Filkov, V. and Skiena, S.: 2004, Heterogeneous data integration with the consensus clustering formalism, *Proceedings of Data Integration in the Life Sciences*, Springer, pp. 110–123.

Flynn, M.: 1972, Some computer organizations and their effectiveness, *IEEE Transactions on Computers* **21**(9), 948–960.

Forestier, G., Wemmert, C. and Gancharski, P.: 2008, Multisource images analysis using collaborative clustering, *EURASIP J. Adv. Signal Process* **2008**, 1–11.

Fred, A. and Jain, A.: 2008, Cluster validation using a probabilistic attributed graph, *Pattern Recognition, 2008. ICPR 2008. 19th International Conference on*, pp. 1 –4.

Fred, A. L. N. and Jain, A. K.: 2005, Combining multiple clusterings using evidence accumulation, *IEEE Transaction on Pattern Analysis and Machine Intelligence* **27**, 835–850.

Freund, Y. and Schapire, R. E.: 1997, A decision-theoretic generalization of on-line learning and an application to boosting,, *Journal of Computer and System Sciences* **55**(1), 119 – 139.

Gionis, A., Indyk, P. and Motwani, R.: 1999, Similarity search in high dimensions via hashing, *Proceedings of the 25th International Conference on Very Large Data Bases* pp. 518–529.

Gllavata, J., Qeli, E. and Freisleben, B.: 2006, Detecting text in videos using fuzzy clustering ensembles, pp. 283 –290.

Gonzalez, E. and Turmo, J.: 2008, Comparing non-parametric ensemble methods for document clustering, *NLDB '08: Proceedings of the 13th international conference on Natural Language and Information Systems*, Springer-Verlag, Berlin, Heidelberg, pp. 245–256.

Gropp, W.: 2002, Mpich2: A new start for mpi implementations, *Lecture Notes in Computer Science* pp. 7–27.

Haveliwala, T., Gionis, A. and Indyk, P.: 2000, Scalable techniques for clustering the web, *WebDB (Informal Proceedings)*, Vol. 129, p. 134.

Hinneburg, A. and Gabriel, H.-H.: 2007, Denclue 2.0: Fast clustering based on kernel density estimation, *In Proc. of 7th International Symposium on Intelligent Data Analysis*, pp. 70–80.

Hongjun, W., Hanhuai, S. and Arindam, B.: 2009, Bayesian cluster ensembles, *Proceedings of the Ninth SIAM International Conference on Data Mining*, SIAM.

Hore, P., Hall, L. O. and Goldgof, D. B.: 2009, A scalable framework for cluster ensembles, *Pattern Recognition* **42**(5), 676–688.

Hu, X. and Yoo, I.: 2004, Cluster ensemble and its applications in gene expression analysis, *APBC '04: Proceedings of the second conference on Asia-Pacific bioinformatics*, Australian Computer Society, Inc., Darlinghurst, Australia, Australia, pp. 297–302.

Hu, X., Zhang, X. and Zhou, X.: 2006, Integration of cluster ensemble and em based text mining for microarray gene cluster identification and annotation, *CIKM '06: Proceedings of the 15th ACM international conference on Information and knowledge management*, ACM, New York, NY, USA, pp. 824–825.

Hubert, L. and Arabie, P.: 1985, Comparing partitions, *Journal of Classification* **2**, 193–218.

Iam-on, N., Boongoen, T. and Garrett, S.: 2010, LCE: a link-based cluster ensemble method for improved gene expression data analysis, *Bioinformatics* **26**(12), 1513–1519.

Indyk, P. and Motwani, R.: 1998, Approximate nearest neighbors: towards removing the curse of dimensionality, *Proceedings of the thirtieth annual ACM symposium on Theory of computing* pp. 604–613.

Jain, A. K.: 2010, Data clustering: 50 years beyond k-means, *Pattern Recognition Letters* **31**(8-18), 651 – 666. Award winning papers from the 19th International Conference on Pattern Recognition (ICPR), 19th International Conference in Pattern Recognition (ICPR).

Kambadur, P., Gregor, D., Lumsdaine, A. and Dharurkar, A.: 2006, Modernizing the c++ interface to mpi, *Lecture Notes in Computer Science* **4192**, 266.

Karypis, G., Aggarwal, R., Kumar, V. and Shekhar, S.: 1999, Multilevel hypergraph partitioning: Applications in vlsi domain, *IEEE transactions on very large scale integration(VLSI) systems* **7**(1), 69–79.

Karypis, G., Han, E.-H. and Kumar, V.: 1999, Chameleon: hierarchical clustering using dynamic modeling, *Computer* **32**(8), 68 –75.

Karypis, G. and Kumar, V.: 1998a, A fast and high quality multilevel scheme for partitioning irregular graphs, *SIAM Journal of Scientific Computing* **20**(1), 359–392.

Karypis, G. and Kumar, V.: 1998b, Multilevel algorithms for multi-constraint graph partitioning, *Supercomputing '98: Proceedings of the 1998 ACM/IEEE conference on Supercomputing (CDROM)*, IEEE Computer Society, Washington, DC, USA, pp. 1–13.

Kirkpatrick, S., Gelatt, C. D. and Vecchi, M. P.: 1983, Optimization by simulated annealing, *Science* **220**, 671–680.

Kittler, J., Hatef, M., Duin, R. P. and Matas, J.: 1998, On combining classifiers, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **20**, 226–239.

Kleinberg, J.: 2002, An impossibility theorem for clustering, *in* S. Becker, S. Thrun and K. Obermayer (eds), *Neural Information Processing Systems 14*, MIT Press, pp. 446–453.

Koga, H., Ishibashi, T. and Watanabe, T.: 2007, Fast agglomerative hierarchical clustering algorithm using locality-sensitive hashing, *Knowledge and Information Systems* **12**(1), 25–53.

Kyrgyzov, I. O., Maitre, H. and Campedel, M.: 2007, A method of clustering combination applied to satellite image analysis, *ICIAP '07: Proceedings of the 14th International Conference on Image Analysis and Processing*, IEEE Computer Society, Washington, DC, USA, pp. 81–86.

Lam, L.: 2000, Classifier combinations: Implementations and theoretical issues, *MCS '00: Proceedings of the First International Workshop on Multiple Classifier Systems*, Springer-Verlag, London, UK, pp. 77–86.

Li, H., Wang, H., Chen, M. and Wang, T.: 2006, Clustering ensemble technique applied in the discovery and diagnosis of brain lesions, *ISDA '06: Proceedings of the Sixth International Conference on Intelligent Systems Design and Applications*, IEEE Computer Society, Washington, DC, USA, pp. 512–520.

Li, Y., Yu, J., Hao, P. and Li, Z.: 2007, Clustering ensembles based on normalized edges, *PAKDD'07: Proceedings of the 11th Pacific-Asia conference on Advances in knowledge discovery and data mining*, Springer-Verlag, Berlin, Heidelberg, pp. 664–671.

Luo, H., Kong, F. and Li, Y.: 2006, Clustering mixed data based on evidence accumulation, *in* X. Li, O. Zaïane and Z. Li (eds), *Advanced Data Mining and Applications*, Vol. 4093 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, pp. 348–355.

Ma, X., Wan, W. and Jiao, L.: 2009, Spectral clustering ensemble for image segmentation, *GEC '09: Proceedings of the first ACM/SIGEVO Summit on Genetic and Evolutionary Computation*, ACM, New York, NY, USA, pp. 415–420.

Mimaroglu, S. and Simovici, D. A.: 2008, Approximate computation of object distances by locality-sensitive hashing, *DMIN*, pp. 714–718.

Mimaroglu, S. and Yagci, A. M.: 2009, A binary method for fast computation of inter and intra cluster similarities for combining multiple clusterings, *ICIS '09: Proceedings of the 2nd International Conference on Interaction Sciences*, ACM, New York, NY, USA, pp. 452–456.

Minaei-bidgoli, B., Topchy, A. and Punch, W. F.: 2004, A comparison of resampling methods for clustering ensembles, *In IC-AI*, pp. 939–945.

Mohammadi, M., Nikanjam, A. and Rahmani, A.: 2008, An evolutionary approach to clustering ensemble, *ICNC '08: Proceedings of the 2008 Fourth International Conference on Natural Computation*, IEEE Computer Society, Washington, DC, USA, pp. 77–82.

Moon, J. and Moser, L.: 1965, On cliques in graphs, *Is. Journal of Mathematics* **3**(1), 23–28.

Ostergard, P. R. J.: 1999, A new algorithm for the maximum-weight clique problem, *Electronic Notes in Discrete Mathematics* **3**, 153–156. 6th Twente Workshop on Graphs and Combinatorial Optimization.

Samudrala, R. and Moult, J.: 1998, A graph-theoretic algorithm for comparative modeling of protein structure, *Journal of Molecular Biology* **279**(1), 287–302.

Sevillano, X., Cobo, G., Alias, F. and Socoro, J. C.: 2006, Robust document clustering by exploiting feature diversity in cluster ensembles, *Procesamiento del Lenguaje Natural*, Vol. 37, Sociedad Espanola para el Procesamiento del Lenguaje Natural, pp. 169–178.

Shi, J. and Malik, J.: 2000, Normalized cuts and image segmentation, *Pattern Analysis and Machine Intelligence, IEEE Transactions on* **22**(8), 888 –905.

Sibson, R.: 1973b, Slink: An optimally efficient algorithm for the single-link cluster method, *The Computer Journal* **16**(1), 30–34.

Silva, L. S. and Scharcanski, J.: 2010, Video segmentation based on motion coherence of particles in a video sequence, *Trans. Img. Proc.* **19**(4), 1036–1049.

Sokal, R. and Rohlf, F.: 1962, The comparison of dendrograms by objective methods, *Taxon* **11**(1), 30–40.

Strehl, A. and Ghosh, J.: 2002, Cluster ensembles - a knowledge reuse framework for combining multiple partitions, *Journal of Machine Learning Research* **3**, 583–617.

Strehl, A., Strehl, E. and Ghosh, J.: 2000, A scalable approach to balanced, high-dimensional clustering of market-baskets, *In Proceedings of HiPC*, Springer, pp. 525–536.

Tansey, W. and Tilevich, E.: 2008, Efficient automated marshaling of c++ data structures for mpi applications., *IPDPS*, IEEE, pp. 1–12.

Thakur, R., Rabenseifner, R. and Gropp, W.: 2005, Optimization of collective communication operations in mpich, *International Journal of High Performance Computing Applications* **19**(1), 49.

Tomita, E., Tanaka, A. and Takahashi, H.: 2006, The worst-case time complexity for generating all maximal cliques and computational experiments, *Theoretical Computer Science* **363**(1), 28–42.

Topchy, A., Jain, A. K. and Punch, W.: 2003, Combining multiple weak clusterings, *Third IEEE International Conference on Data Mining, 2003. ICDM 2003*, pp. 331–338.

Topchy, A., Jain, A. K. and Punch, W.: 2005, Clustering ensembles: Models of consensus and weak partitions, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **27**, 1866–1881.

Topchy, A., Jain, A. and Punch, W.: 2004, A mixture model for clustering ensembles, *Proceedings of the 2004 SIAM International Conference on Data Mining*, SIAM.

Topchy, A., Law, M., Jain, A. and Fred, A.: 2004, Analysis of consensus partition in cluster ensemble, *Data Mining, 2004. ICDM '04. Fourth IEEE International Conference on*, pp. 225 – 232.

Tu, B., Fan, J., Zhan, J. and Zhao, X.: 2009, Performance analysis and optimization of mpi collective operations on multi-core clusters, *The Journal of Supercomputing* pp. 1–22.

Tumer, K. and Agogino, A. K.: 2008, Ensemble clustering with voting active clusters, *Pattern Recognition Letters* **29**(14), 1947–1953.

Vendramin, L., Campello, R. J. G. B. and Hruschka, E. R.: 2009, On the comparison of relative clustering validity criteria, *SDM*, pp. 733–744.

Wolfgang, v. d. G., Koeppen, M. and Dimitriadou, E.: 2000, Robust clustering by evolutionary computation, *Proc. 5th Online World Conference on Soft Computing in Industrial Applications (WSC5)*.

Xu, S., Lu, Z. and Gu, G.: 2008, An efficient spectral method for document cluster ensemble, pp. 808 –813.

Yang, Y., Kamel, M. and Jin, F.: 2006, Clustering ensemble using ant and art, *Swarm Intelligence in Data Mining*, Springer, pp. 243–264.

Yu, Z., Wong, H.-S. and Wang, H.: 2007, Graph-based consensus clustering for class discovery from gene expression data, *Bioinformatics* **23**(21), 2888–2896.

Zhang, X., Jiao, L., Liu, F., Bo, L. and Gong, M.: 2008, Spectral clustering ensemble applied to sar image segmentation, *Geoscience and Remote Sensing, IEEE Transactions on* **46**(7), 2126 –2136.

Zhang, Z., Cheng, H., Zhang, S., Chen, W. and Fang, Q.: 2008, Clustering aggregation based on genetic algorithm for documents clustering, pp. 3156 –3161.

Other References

Asuncion, A. and Newman, D.: 2007, Uci machine learning repository.

SHORT CV

Name Surname	:	Arif Murat YAĞCI
Address	:	Bahçeşehir Üniversitesi Mühendislik Fakültesi Çırağan Caddesi 34353 Beşiktaş / ISTANBUL
Languages	:	Turkish (native), English (fluent), German (intermediate)
B.S.	:	Istanbul Technical University (ITU)
M.S.	:	Bahcesehir University
Institute	:	The Graduate School of Natural and Applied Sciences
Program	:	Computer Engineering
Publications	:	S. Mimaroglu, M. Yagci, 2010, CLICOM: Cliques for Combining Multiple Clusterings, *Journal*, submitted. S. Mimaroglu, M. Yagci, 2010, Efficient Fitness Function for Combining Multiple Clusterings by Evolutionary Methods, *Journal*, submitted. S. Mimaroglu, M. Yagci, D.A. Simovici, 2010, Approximate Computing of Distances by Random Hashing, *Journal*, submitted. S. Mimaroglu, A. M. Yagci, 2009, A Binary Method for Fast Computation of Inter and Intra Cluster Similarities for Combining Multiple Clusterings, *in Proc. ACM, ICCIT, Korea*.
Work Experience	:	Bahcesehir University Computer Engineering Department *Research and Teaching Assistant* (Istanbul, 2008 - today) Industry *Research and Development Engineer, Systems Engineer, Software Developer* (Turkey, CIS, China, 2001 - 2008)