Multi-View Hierarchical Clustering using Optimal Transport

DISSERTATION SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Technology
in
Computer Science

by

Sohan Ghosh

[Roll No: CS1910]

under the guidance of

Dr. Swagatam Das

Associate Professor
Electronics and Communication Sciences Unit

Indian Statistical Institute
Kolkata-700108, India

July, 2021

To my friends and family

CERTIFICATE

This is to certify that the dissertation entitled "**Multi-View Hierarchical Clustering using Optimal Transport**" submitted by **Sohan Ghosh** to Indian Statistical Institute, Kolkata, in partial fulfillment for the award of the degree of **Master of Technology in Computer Science** is a bonafide record of work carried out by him under my supervision and guidance. The dissertation has fulfilled all the requirements as per the regulations of the institute and, in my opinion, has reached the standard needed for submission.

Swagatam Das
Associate Professor,
Electronics and Communication Sciences Unit,
Indian Statistical Institute,
Kolkata-700108, India

Acknowledgements

I would like to express my highest gratitude to my advisor, *Dr. Swagatam Das*, Associate Professor, Electronics and Communication Sciences Unit, Indian Statistical Institute, Kolkata, for his guidance and continuous encouragement. Without his support this work would not have been possible.

My utmost thanks goes to all the teachers of Indian Statistical Institute for their suggestions and discussions whenever I have needed them, which has undoubtedly helped me to improve my research work.

I would also like to thank all of my friends for their help and support. In particular, I would like to express my appreciation to my classmate Abhirup Gupta for his time and valuable insights during the formulation of this work. And Last but not least, I am very much thankful to my parents and family for their everlasting support.

Sohan Ghosh
Indian Statistical Institute
Kolkata - 700108 , India

Abstract

With the growing availability of multi-view data, development of multi-view clustering algorithms has gained prominence among researchers. However, most of these algorithms are either based on subspace, graph or spectral clustering techniques, with very few works done in terms of hierarchical clustering. In this work, we aim to develop a Multi-View Agglomerative Hierarchical Clustering algorithm which uses Optimal Transport (OT) for calculating distances between clusters. This takes into consideration the entire data distribution of the clusters, unlike traditional single or complete linkage techniques.

When incorporated naively in hierarchical clustering, OT imposes high time complexity. To tackle this we have a Nearest Neighbor Agglomeration (NNA) step which merges multiple clusters in each iteration using chains of first nearest neighbors. This subsequently results in very few iterations and we show that incorporating OT in this setup still leads to relatively low time complexity. Before NNA we have a Cosine or Euclidean Distance Integration (CDI/EDI) step, which essentially calculates the distance between two data samples as the average over their distances in all the views.

Extensive experiments performed on both single-view and multi-view datasets illustrate the efficiency of our algorithm when compared to other state-of-the-art single-view hierarchical clustering and multi-view clustering algorithms respectively.

Keywords: Multi-View Data, Multi-View Clustering, Hierarchical Clustering, Optimal Transport

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Introduction

Data is one of the most valuable commodities in modern-day life. This is because of its abundance with the advent of modern technologies and big data. With this, multi-view data [1] is available to us aplenty and in turn multi-view clustering has become a research topic of significant interest among the scientific community. With the different forms of information obtained from multi-view data, this is really of no surprise.

Imagine a video stream. The image frames and speech can be considered as different views which individually are capable of conveying the desired information. However, many of us would agree that simultaneously listening to the speech and watching the image frames can lead to an improved understanding of the information that the video is trying to convey. Similarly the same word, say "Hello" can be represented in multiple languages [1]. Each of these languages can essentially be thought of as separate views. Chapter 2 contains more examples and detailed analysis of multi-view data and in general multi-view learning.

Many algorithms have been proposed for clustering multi-view data in recent years. Arguably the most widely used category among them is the Subspace Clustering based algorithms. The naive or typical approach for this category is to learn a final affinity matrix which is formed by combining the individual affinity matrices of each view. Then Spectral Clustering [5] is applied to get the partitions. Several variations have also been proposed in recent years. MVSC [6] uses a cluster indicator matrix that is treated to be common with respect to all the views while attempting to learn a graph representation for all views. LMVSC [7] uses the concept of anchor graphs to reduce the time complexity to linear. Using the information that is complementary to all the different views by finding a latent representation of the views under consideration has been proposed in LMSC [3].

Also used often are Spectral Clustering based approaches. [8] initially forms probability matrices corresponding to the different views and then derives a combined probability matrix shared among those views. This is done using sparse and low-rank representations. GMC [9] uses a mutual reinforcement strategy to learn the graph matrix of each view and the unified graph matrix. Apart from these two popular categories, several other well accepted works have been proposed as well [10][11][12].

Due to the large amount of data inherently existing in a hierarchical structure,

Hierarchical Clustering, without a doubt, is one of the most widely used clustering methods as far as single-view data in concerned. A lot of new algorithms have been developed recently [13][14][15][16][17][18][19]. FINCH [20] uses chains of first nearest neighbors to merge multiple clusters in each step, thus leading to very few number of iterations and low time complexity. Typically one would expect hierarchical clustering to be equally popular with multi-view data. However, it is a bit surprising to find that there is not much literature available on multi-view hierarchical clustering. This is possibly because of the question as to how the distances of samples in different views can be aggregated during the merging or division process in agglomerative or divisive hierarchical clustering respectively. Probably the first notable multi-view hierarchical clustering method was proposed by [21] where they used dendograms of different views to derive a combined cophenetic distance matrix, from which a final aggregate dendogram can be formed. Recently [4] has proposed MHC, a multi-view agglomerative hierarchical clustering scheme which essentially considers the average over the cosine distances of all the views as the distance in the latent representation, which is then used for merging of clusters based on chains of first nearest neighbors. Using Optimal Transport (OT) [22] to pick the two clusters to merge in an iteration was introduced in HC-OT [23]. This ensures that the entire data distribution of the clusters is considered while making the decision of merging. However HC-OT is restricted to single-view domain and is also very time consuming.

This brings us to our proposed Multi-View Agglomerative Hierarchical Clustering technique (MVHC-OT) which essentially incorporates OT in the setup suggested by [4]. This allows us to leverage the utilities of OT over traditional measures like single-linkage or complete-linkage as in the single-view scenario described in [23]. At the same time the setup of [4] ensures that in spite of incorporating OT, we are able to maintain a relatively low time complexity.

1.2 Our Contributions

The main contributions of our work can be summarized as follows:

- We propose a Multi-View Agglomerative Hierarchical Clustering Algorithm which uses Optimal Transport (OT) to decide which clusters to merge in each step. Using OT helps our method to compute the distance between the concerned clusters while keeping in mind their entire data distribution, unlike other traditional methods.

- Our Multi-View Hierarchical Clustering Algorithm merges multiple clusters and produces inherently meaningful clusters in each step. This results in very few iterations even for large datasets which in turn leads to a relatively low time complexity despite the incorporation of OT.

- Elaborate experiments have been performed on real-life multi-view datasets to compare the performance of our proposed algorithm with state-of-the-art multi-view clustering methods. In addition, we provide results on some real-life single-view datasets as well, showing that our proposed multi-view clustering algorithm generalizes well for single-view data too.

1.3 Thesis Outline

The rest of this thesis is outlined as follows. Chapter 2 contains the Preliminary topics related to our method. Chapter 3 summarizes the Related Works done in Hierarchical Clustering, both in single-view and multi-view domain. Chapter 4 discusses our proposed Multi-View Hierarchical Clustering Algorithm and Chapter 5 gives its detailed Performance Analysis. Finally Chapter 6 concludes our work along with some future directions.

Chapter 2

Preliminaries

2.1 Notations

The notations used in this work are stated in Table 2.1. We follow these notations throughout this work, unless otherwise mentioned.

Notations	Description
n	Number of samples in dataset
d	Number of features
$X = (x_{ij}) \in \mathbb{R}^{n \times d}$	Single-view dataset in matrix format
v	Number of views
d_k	Number of features in k-th view
$X^{(k)} = (x_{ij}^{(k)}) \in \mathbb{R}^{n \times d_k}$	k-th view of the dataset in matrix format
K	Number of clusters
μ, ν	Probability distributions
$C = (c_{ij}) \in \mathbb{R}^{m \times n}$	Cost matrix of Optimal Transport
$\gamma = (\gamma_{ij}) \in \mathbb{R}^{m \times n}$	Transportation matrix of Optimal Transport
$a = [a_1, ..., a_m]^T, b = [b_1, ..., b_n]^T$	Discrete probability masses
λ	Entropic regularization constant in Optimal Transport
$d^{\lambda}(\mu, \nu)$	Sinkhorn distance between μ and ν
$S_{\lambda}(C_1, C_2)$	Sinkhorn distance between clusters C_1 and C_2
$D = (d_{ij}) \in \mathbb{R}^{p \times p}$	Single-view distance matrix when there are p clusters left to be merged
$D^{(k)} = (d_{ij}^{(k)}) \in \mathbb{R}^{p \times p}$	Distance matrix of k-th view when there are p clusters left to be merged
$D^* = (d_{ij}^*) \in \mathbb{R}^{p \times p}$	Essential Distance matrix when there are p clusters left to be merged
$P^{(k)}$	Orthogonal matrix of the k-th view
H	Latent representation of multiple views
$\mathcal{R} = \{R_1, R_2, ...\}$	Partitions obtained during each step of hierarchical clustering
$\lvert R_k \rvert$	Number of clusters in the k-th partition
$\mathcal{G}$	Nearest neighbor adjacency graph matrix

Table 2.1: Notations used

2.2 Multi-View Learning

2.2.1 Motivation

From what can we identify a human being? His/her face, signature, voice tone, finger prints, dental records etc can each help in identification of a person on its own [1]. However there always remains a possibility that there is some noise or ambiguity in the above features collected from different sources which are detrimental to the recognition process. In such scenarios it is quite natural to think that a combination of the above features will aid in the identification task better. This is the motivation behind Multi-View Learning.

2.2.2 Multi-View Data

In our day-to-day life, we encounter multi-view data every now and then. Different newspapers report the same incident in different languages and from different perspectives. Different features like color, texture, SIFT, Gabor etc describe the same image. Images shared in a website can be described by the image itself and also by the text in the regions surrounding that image. All these can be thought of as examples of multi-view data.

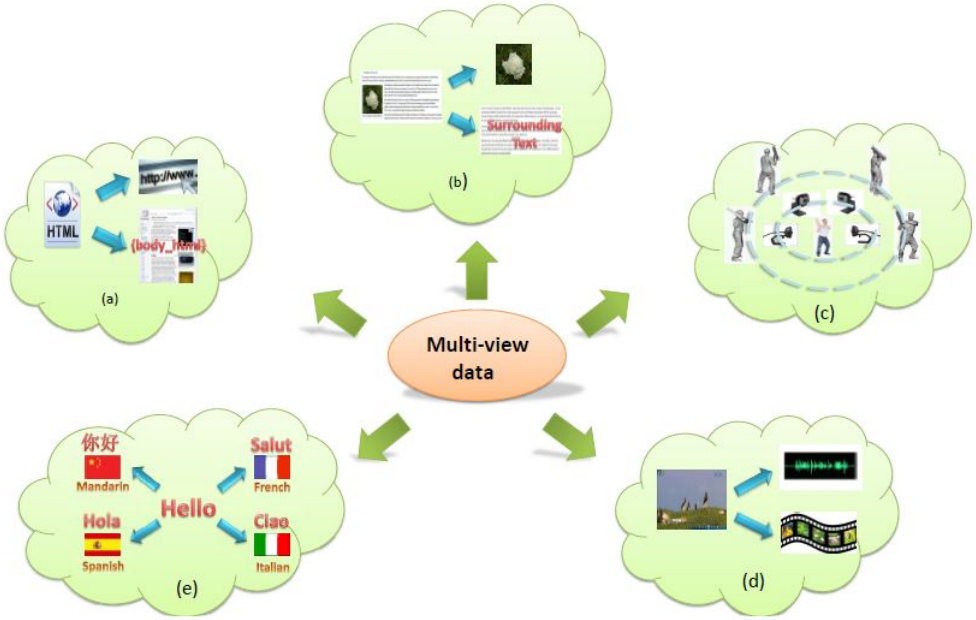


Figure 2.1: Examples of multi-view data (Image source: [1])

As we approach the age of big data where data is available in abundance to us, different sources have emerged which provide data about the same entity; thus providing different perspectives at the same time. The term '*multi-view data*' refers to these data and the sources are termed as '*views*' [2]. Although these data have properties that are heterogeneous to each other, there exists some connection among them which can prove to be fruitful in the learning task [24]. Exploiting these connections for learning is the real aim of multi-view learning.

2.2.3 Multi-View Clustering Principles

This leads us to Multi-View Clustering (MvC) [2][25] which has drawn increasing attention of the researchers in recent times. MvC operates on the two principles: Complementary and Consensus.

1. **Complementary Principle:** Although each single view is efficient to some extent with respect to a particular kind of knowledge extraction task, separate views provide information that is complementary to one another and hence they act as much better descriptors for the data objects as a whole. So deploying multiple views is essential.

2. **Consensus Principle:** Since the different views are describing the same object, the consistency across all the views should be maximized. This is the simple goal of the consensus principle. [26] shows that the generalization error of the classifiers on two independent views can be considered to be an upper bound for the classification error on either view. Thus, a maximization in the consensus among the two views should in turn result in the minimization of error of each view.

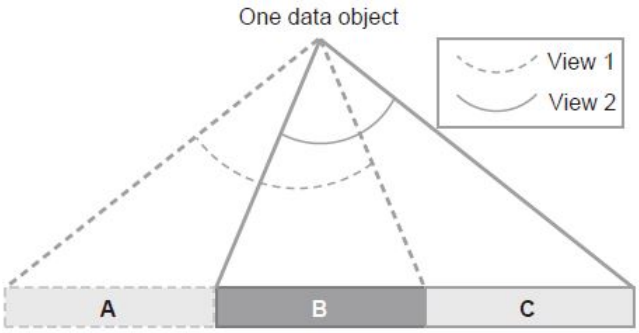


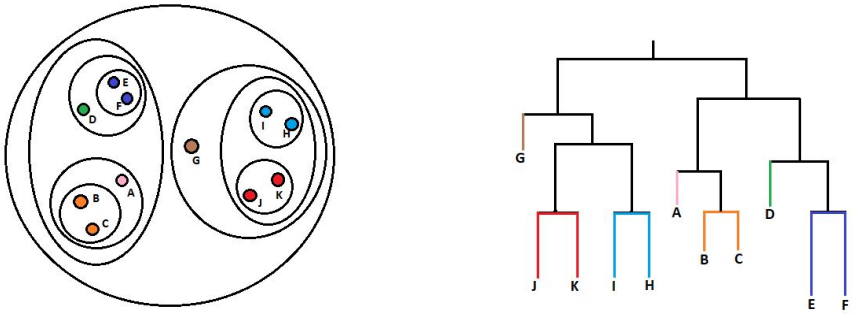
Figure 2.2: Example of complementary and consensus principles (Image source: [2])

[2] illustrates the complementary and consensus principles using Figure 2.2 in the following way. A data object expressed in the latent space has two inherent views. Now, part A and part C are present exclusively in View 1 and View 2 respectively. These represent the Complementary information of the two views. On the other hand, part B is shared by both the two views. This represents the Consensus of the two views.

Both these principles form the crux of MvC problem. However they are in a way contradictory to each other. Hence the true challenge of MvC lies in the proper balance between these two principles.

2.3 Hierarchical Clustering

Clustering [27] in one of the core facets of machine learning, where given a datatset we aim to 'cluster' or group the data samples together into different sets based on their properties or features. This kind of learning problem is *unsupervised* in nature as we are not provided with any training labels with which we can train our model



(a) Partitions obtained (b) Dendogram obtained

Figure 2.3: Example of hierarchical clustering

and then use it to predict the class/group of samples, as is the case for *supervised learning*.

Clustering can be very broadly divided into two categories:

1. **Partitional Clustering [28]:** Here the dataset is partitioned into a set of clusters such that every data point is a part of one and only one cluster. A very common example of this kind of clustering technique is the K-means algorithm.

2. **Hierarchical Clustering [29][30]:** Here the dataset is partitioned into different clusters such that a data point can be a part of more than one clusters. This type of data is naturally observed in nature in many form and its quite obvious that we wish to have a clustering structure resembling it. A tree like structure called the *dendogram* is used to represent this hierarchical structure. Figure 2.3 provides an illustration of this. This kind of clustering is of primary concern to us in this thesis and hence let us delve a bit more elaborately into this.

Hierarchical Clustering can be mainly divided into categories:

1. **Divisive:** This is a *top-down* approach. The idea is to consider the entire dataset as a single cluster first. Then based on some metric, this cluster is divided into two clusters. In the next step, one of the existing clusters is again chosen and divided into two parts. This precess goes on until we are left with only one data point in each cluster. Notice that this process enables each data point to be a part of more than one cluster based on the clustering level.

2. **Agglomerative:** Contrary to the Divisive method, this approach is *bottom-up*. We start with each data point as individual, separate clusters. Then in each iteration, based on some metric, we merge two chosen clusters into a single cluster. We stop when all data points belong to one single cluster. As evident, similar to the divisive approach, we are once again able to cluster each data point into more than one partition based on the clustering level or step.

Our proposed approach depicted in this thesis in Chapter 4 falls under the Agglomerative Hierarchical Clustering framework.

2.4 Optimal Transport

2.4.1 Motivation

Optimal Transport (OT) [22] is essentially a distance measure between probability distributions. Its history originates back to the year 1781 when Monge [31] first introduced it to solve the problem of resource allocation. It was described through an example of moving a pile of earth from a source location to another destination location. The idea is that there is a pile of earth with a particular shape at a particular location. At some distance there is a hole of some shape. A person has to use a shovel and manually shift the pile of earth to put it in the destination hole. The main motive is to reduce the total cost of transportation. Hence OT distance is often referred to as earth-mover's distance as well. This is analogous to using OT to compare two probability distributions (the pile of sand in the source location and the hole in the destination location of same volume can be thought of as two probability distributions).

2.4.2 Monge Problem

Let us formally state the Monge Problem [31]. First, let us assume Ω_s and Ω_t to be two Polish Spaces [32]. Then, consider $c : \Omega_s \times \Omega_t \rightarrow [0, \infty]$ to be a Borel-measurable cost function. Assuming μ to be the probability measure on Ω_s and ν to be the probability measure on Ω_t, a transport map T is defined from Ω_s to Ω_t.

$$\inf_{T:T_{\#}\mu=\nu} \int_{\Omega_s} c(x, T(x)) d\mu(x) \tag{2.1}$$

Monge treats OT problem as finding a T such that Equation (2.1) is minimized under the condition that $T_{\#}\mu = \nu$. Here $\mu(T^{-1}(B)) = \nu(B)$, $\forall B \in \nu$, ie $T_{\#}\mu$ must push-forward μ by T towards ν.

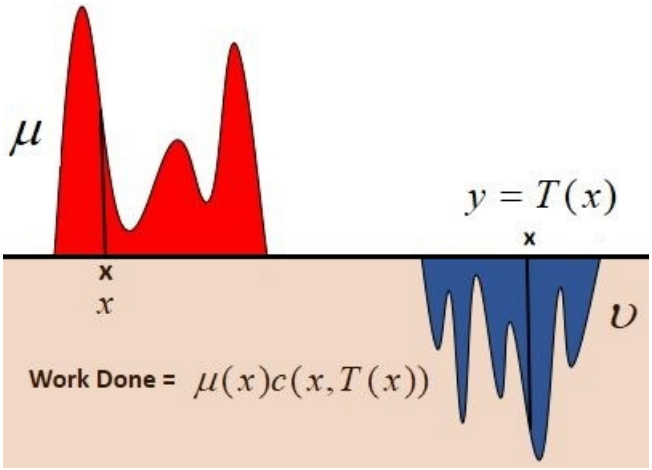


Figure 2.4: Illustration of Monge's earth mover's distance

2.4.3 Kantorovich Problem

Issue with Monge's Formulation

The constraint imposed on the Monge Problem makes it ill-posed at times, since there may not be any valid mapping which satisfies this constraint of $T_{\#}\mu = \nu$. To understand the issue, without going into mathematical terms, let us once again go back to the analogy of moving a pile of earth from a source location to a destination location. The above constraint simply means that every particle of sand located at the same position in the source location must be shifted to the same position in the destination location. It is quite obvious that this may not be at all possible in a lot of cases and depends on the shape of the hole to be filled. Here lies the issue with Monge's Formulation.

Kantorovich Relaxation

To solve this problem, Kantorovich proposed his formulation of the OT problem in 1942 [33]. Assume Γ to be the set of all measure couples between μ and ν. Given this, Kantorovich proposed to solve Equation (2.2) such that $\gamma \in \Gamma(\mu, \nu)$.

$$\inf_{\gamma \in \Gamma(\mu,\nu)} \int_{\Omega_s \times \Omega_t} c(x, y) d\gamma(x, y) \tag{2.2}$$

[32] shows that under certain conditions of regularity, Equation (2.2) is always solvable.

2.4.4 Discrete OT

If we shift our focus to only discrete domain, then the Kantorovich Problem of Equation (2.2) can be considered as a Linear Programming Problem. Consider μ and ν to be two discrete probability distributions on $\mathbb{R}^d$ (note that these were in the continuous domain for Monge and Kantorovich Problem). Let us assume δ_{z_i} to be the Dirac delta function at point $z_i \in \mathbb{R}^d$. If a_i and b_i are probability masses at points x_i and y_j respectively, such that $\sum_{i=1}^{m} a_i = 1$ and $\sum_{j=1}^{n} b_j = 1$, then we can define $\mu = \sum_{i=1}^{m} a_i \delta_{x_i}$ and $\nu = \sum_{j=1}^{n} b_j \delta_{y_j}$. Given this setup, assuming c_{ij} to be the total cost of transporting from point x_i to y_j, the discrete version of the Kantorovich Problem is given by Equation (2.3).

$$\min_{\gamma} \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij}\gamma_{ij} \text{ such that } \gamma 1_n = a, \gamma^T 1_m = b \tag{2.3}$$

where $a = [a_1, ..., a_m]^T$, $b = [b_1, ..., b_n]^T$.

2.4.5 Sinkhorn Distance

Issues with Traditional Discrete OT

[34] discusses elaborately the issues with the OT formulation described in Section 2.4.4. We briefly mention them as follows:

i) If $m = n$ in the discrete setup, then the minimum time taken by any flow solver available for use in practice is $\mathcal{O}(n^3 logn)$. This is clearly is not suitable for Machine Learning problems which require fast processing. This is the prime reason why despite having its origin in the 18th century, OT has not been exploited enough by the Machine Learning community.

ii) The solution is not always unique for the discrete OT problem. Machine Learning applications desire a more stable solution.

Solution using Entropic Regularization

[34] proposed to use Sinkhorn's Algorithm [35] to solve Equation (2.4) such that only fixed point iterations are used.

$$\min_{\gamma}\{\sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}\gamma_{ij} + \lambda\sum_{i=1}^{m}\sum_{j=1}^{n}\gamma_{ij}log(\gamma_{ij})\} \text{ such that } \gamma 1_n = a, \gamma^T 1_m = b \qquad (2.4)$$

With respect to the cost matrix C, we denote the required solution of Equation (2.4) as $d_C^{\lambda}(\mu, \nu)$, the Sinkhorn Distance between μ and ν. For details on how to come up with a solution for Equation (2.4), readers are suggested to refer to Appendix A.

Solving the above equation is found to give a time complexity of $\mathcal{O}(mn)$. Also, the intuition behind adding the regularization term is as follows. The entropy term $-\sum_{i=1}^{m}\sum_{j=1}^{n}\gamma_{ij}log(\gamma_{ij})$ is strongly concave in nature. Subtracting it from the original discrete OT objective function makes it convex in nature, thus making the solution to Equation (2.4) stable.

Chapter 3

Related Works

In this Chapter we focus on some algorithms which hold key components which we have taken inspiration from in our proposed multi-view hierarchical clustering method.

3.1 Hierarchical Clustering using Optimal Transport (HC-OT)

In this section we discuss the Hierarchical Clustering using Optimal Transport (HC-OT) algorithm [23] with uses the concept of OT to decide which clusters to merge during agglomerative hierarchical clustering.

3.1.1 Formulation

The main motivation behind using OT instead of the traditional single or complete linkage schemes is to consider the entire distribution of the clusters, which is not done in the traditional schemes (only the maximum and minimum distance between the two concerned clusters are taken into consideration for complete and single linkage schemes respectively). The most important thing here is to define a measure in terms of OT for the distance between two clusters.

Consider C_1 and C_2 to be two clusters. Now, assuming that x_i is the i-th point of C_1, $i = 1, ..., |C_1|$ and y_j is the j-th point of C_2, $j = 1, ..., |C_2|$, the distribution of the data points in the two clusters can be given as $\mu = \sum_{i=1}^{|C_1|} \frac{1}{|C_1|}\delta_{x_i}$ and $\nu = \sum_{j=1}^{|C_2|} \frac{1}{|C_2|}\delta_{y_j}$. For $\lambda > 0$, if the distance between x_i and y_j is taken to be c_{ij}, the Sinkhorn distance between μ and ν can also be considered to be the Sinkhorn distance between the two clusters C_1 and C_2, ie. $S_\lambda(C_1, C_2) = d_C^\lambda(\mu, \nu)$.

However, it is to be noted that only if $|C_1|$ and $|C_2|$ are large enough, can we come up with the above approximation. [23] uses Sinkhorn distance as a distance measure between two clusters under consideration when they both have at least $n/10$ data points. Hence, in the situation where this condition is not satisfied, the traditional single or complete linkage is used instead of OT as the distance between the respective clusters.

The HC-OT algorithm as per [23] is shown in Algorithm 1.

3.1.2 Observations

[23] has presented the performance of HC-OT on two synthetic as well as several real-life datasets and done an elaborate comparative analysis with other state-of-the-art hierarchical clustering algorithms to show the improvement in performance while using HC-OT in most of the cases.

However, the biggest issue faced by this algorithm is the increase in time complexity compared to the naive agglomerative single linkage scheme which takes $\mathcal{O}(n^3)$. This is due to the incorporation of OT in distance computation which takes $\mathcal{O}(n^2)$ for each computation. This begs the questions. Can we incorporate OT into the hierarchical clustering framework with an improved time complexity? Also, HC-OT is essentially a single-view clustering algorithm. How about having a multi-view hierarchical clustering framework which incorporates OT? In addition, HC-OT works on the prior knowledge of the actual or required number of clusters in the data. Can it be modified such that even without the knowledge of the actual number of clusters, it can give a clustering result with approximately the actual number of clusters or may give a clustering that somehow has some inherent meaning?

Algorithm 1: HC-OT Algorithm

Input : Distance matrix $D \in \mathbb{R}^{n \times n}$, λ
Output: Clustering dendogram

Initialize all data points as individual clusters;
$L(0) = 0$;
$m = 0$;
while *number of clusters greater than 1* **do**
  Find cluster pair (p) and (q) such that $d[(p),(q)] = min_{(r),(s)} d[(r),(s)]$, $\forall((r),(s)) \in D$;
  Combine the (p) and (q) clusters into one cluster to form the new clustering level;
  $L(m) = d[(p),(q)]$;
  $m = m + 1$;
  Update D by eliminating the columns and rows corresponding to the (p) and (q) clusters, and then including a new column and row for the new cluster;
  The distance between the new cluster indicated as (p,q), and a former cluster (r) can be expressed as:

$$d[(r),(p,q)] = \begin{cases} S_\lambda((r),(p,q)), & \text{if } |(r)|,\ |(p,q)| > \frac{n}{10} \\ min\{d[(r),(p)], d[(r),(q)]\}, & \text{otherwise} \end{cases};$$

end

3.2 Hierarchical Clustering using First Nearest Neighbors (FINCH)

In this section we take an elaborate look into [20], which presents an agglomerative hierarchical clustering algorithm called FINCH. FINCH, unlike HC-OT [23] does not require any hyperparameter, threshold or actual number of clusters. The main intuition is that large chains in the dataset can be formed only by connecting

data points via their first nearest neighbors, which essentially leads to large groupings/clusters among the data in a significantly small number of iterations. Also the time complexity is shown to be very low and as a result can be scaled to large datasets.

3.2.1 Formulation

FINCH Clustering Equation

Let κ_i^1 be the first nearest neighbor of data point i. Then, if the first nearest neighbor corresponding to every data point is known, [20] builds an adjacency link matrix defined as follows.

$$\mathcal{G}(i,j) = \begin{cases} 1, \text{ if } j = \kappa_i^1 \text{ or } \kappa_j^1 = i \text{ or } \kappa_i^1 = \kappa_j^1 \\ 0, \text{ otherwise} \end{cases} \tag{3.1}$$

The above equations essentially lead to a symmetric sparse matrix $\mathcal{G}$, whose strongly connected components are the cluster partitions. The condition $j = \kappa_i^1$ joins a point i to its first nearest neighbor. $\kappa_j^1 = i$ ensures symmetry. And $\kappa_i^1 = \kappa_j^1$ joins the points (i, j) which have the same neighbor.

3.2.2 The Algorithm

Finding the connected components from the adjacency matrix $\mathcal{G}$ after applying Equation (3.1) gives a flat clustering of the data in the first iteration. The simple idea is to recursively follow this same procedure as in the typical agglomerative hierarchical clustering method repeatedly. Notice that unlike traditional agglomerative approaches which merge only two clusters in each step, FINCH merges a large number of clusters in each iteration. As a result convergence is achieved in only a few number of iterations unlike other methods which take $n - 1$ steps. Results in [20] show that even for large datasets having about 1000 to 8 millions samples, FINCH converges in only 4-10 iterations. In addition the clusters provided in each iteration is meaningful in its own inherent way. The number of iterations and also the number of clusters returned in each step bear no direct relationship with the number of samples n.

The FINCH Algorithm as per [20] is given in Algorithm 2 and Algorithm 3. Algorithm 2 describes FINCH procedure when the actual number of clusters is not provided to the algorithm as prior input data. FINCH also works if we require a clustering with a particular number of clusters. This mode of FINCH is captured in Algorithm 3.

As can be seen in Algorithm 2, the first neighbors can be computed through fast approximation algorithms like k-d tree [36] instead of computing an entire distance matrix. This essentially reduces the time complexity of FINCH in comparison with other state-of-the-art hierarchical clustering algorithms by taking only $\mathcal{O}(nlogn)$ time. In contrary, computing distances in the usual manner would have taken $\mathcal{O}(n^2logn)$.

Algorithm 3 uses the result of Algorithm 2 to give us a clustering with the required number of clusters. The partition containing the number of clusters just

greater than the required number of clusters is chosen. Then we proceed in an iterative manner, merging only two clusters at a time until we reach the required number of clusters.

Algorithm 2: FINCH Algorithm

Input : Data matrix $X \in \mathbb{R}^{n \times d}$

Output: Set of Partitions $\mathcal{R} = \{R_1, R_2, ..., R_L\}$ where each partition $R_i = \{C_1, C_2, ..., C_{|R_i|}\}$ *st* $|R_i| > |R_{i+1}|$ $\forall i \in \mathcal{R}$ is a valid clustering of X

Compute $\kappa^1 \in \mathbb{R}^{n \times 1}$ using exact distance or k-d tree;
Compute first partition R_1 containing $|R_1|$ clusters using κ^1 via Equation (3.1);
while $|R_i| \geq 2$ **do**
    Given X and its partition R_i, compute cluster means and use these means as new data points to form new data matrix $W \in \mathbb{R}^{|R_i| \times d}$;
    Compute first neighbors integer vector $\kappa^1 \in \mathbb{R}^{|R_i| \times 1}$ of points in W;
    Given κ^1 get partition R_W of R_i via Equation (3.1), where $R_W \supseteq R_i$;
    if $|R_W| = 1$ **then**
        break;
    else
        Update cluster labels in $R_i : R_W \rightarrow R_i$;
    end
end

Algorithm 3: FINCH Algorithm: Required Number of Clusters Mode

Input : Data matrix $X \in \mathbb{R}^{n \times d}$, a partition R_i from the result of Algorithm 2

Output: Clustering R_K which contains the required K number of clusters

for *steps* $= |R_i| - |R_K|$ **do**
    Using X and its partition R_i, calculate cluster means to form a new data matrix $W \in \mathbb{R}^{|R_i| \times d}$;
    Compute first neighbors integer vector $\kappa^1 \in \mathbb{R}^{|R_i| \times 1}$ of points in W;
    Calculate adjacency matrix $\mathcal{G}$ using κ^1 via Equation (3.1);
    Find d_{min}, the minimum distance between all cluster pairs (k, l) for which $\mathcal{G}(k, l) = 1$;
    Keep the symmetric link in $\mathcal{G}$ for the cluster pair (i, j) corresponding to d_{min}, and set the rest to zero;
    Modify the labels of the clusters in R_i: Merge corresponding (i, j) clusters in R_i;
end

3.2.3 Observations

The general approach for agglomerative hierarchical clustering is as follows. A distance matrix is built from the data samples, which is then used for merging two clusters having least distance in each step. Also, we go on updating the distance matrix with each iteration. This not only requires extra space, but also access-

ing the distance matrix in each step takes time. FINCH essentially avoids these computational issues by maintaining only the first nearest neighbor relations. Also, [20] compares the performance of FINCH with some state-of-the-art clustering techniques to show its superiority.

In particular we can observe no issues with this algorithm. And it is better than HC-OT as far as time complexity is concerned. Also, the fact that it can work without knowing the actual number of clusters in the dataset is an added advantage. However, like HC-OT, FINCH is essentially a single-view clustering technique. Hence, an extension to multi-view framework can be an obvious direction. Also, what happens with respect to time complexity and performance if we can somehow incorporate OT into FINCH? Section 4.1 of Chapter 4 discusses this possibility with our proposed FINCH-OT Algorithm.

3.3 Past works on Multi-View Hierarchical Clustering

Considering the amount of data inherently available in the form of hierarchies, it is surprising that not many of the recent works on multi-view clustering are based on the hierarchical clustering approach. In the following subsections we elaborately describe two significant works in this domain.

3.3.1 Multi-View Agglomerative Clustering Algorithm based on Ensemble of Dendograms

In this section we take a look at [21], which, to the best of our knowledge, was the first proposal for a hierarchical clustering approach for multi-view data. This approach, called the Multi-View Agglomerative Clustering based on Ensemble of Partitions of Different Views combines the dendograms obtained after performing agglomerative clustering on every view independently. An intermediate matrix representation for dendograms is used instead of combining them directly to improve the performance with respect to time taken.

More specifically, the concept of cophentic distance [37][38][39] is used for combining the individual dendograms of each view. The cophenetic distance between two data samples can be defined as the height of the dendogram at which those two samples are first joined or merged into a single cluster. Essentially a cophenetic description matrix is built from each dendogram which consists of the cophenetic distances among the different samples in the dataset. Before proceding futher, these matrices are normalized so that the values lie between 0 and 1.

After forming these normalized matrices they can be aggregated in some suitable way like taking average or maximum. Then a combined final dendogram for all the views is generated from the aggregated cophenetic matrix. Algorithm 4 depicts the algorithm as described by [21].

Observations

This method is significant for being one of the first to attempt a hierarchical clustering approach for multi-view clustering. But, the fact that it individually performs

agglomerative clustering on each view has its disadvantage with respect to time complexity.

Algorithm 4: Multi-View Clustering based on Ensemble of Dendograms

Input : $\{X^{(i)}\}_{i=1}^{v} \in \mathbb{R}^{n \times d}$, $\{D^{(i)}\}_{i=1}^{v} \in \mathbb{R}^{n \times n}$
Output: Combined dendogram for all v views

```
Initialize cluster C_i = x_i, ∀i = 1, ..., n, where x_k is the k-th data sample;
for p = 1, ..., v do
    for q = 1, ..., n do
        (C_i, C_j) = Closest cluster pair;
        Merge C_i and C_j;
    end
end
DD^(i) = Cophenetic matrix of the i-th view, ∀i = 1, ..., v;
NDD^(i) = Normalized Cophenetic matrix of the i-th view, ∀i = 1, ..., v;
Combined Descriptor = aggregation of all the dendogram descriptors in
 appropriate manner;
Final Dendogram = Use the Combined Descriptor to form the links and
 thus build the tree;
```

3.3.2 Multi-View Hierarchical Clustering based on Nearest Neighbors

We devote this section for discussing about [4], which presents Multi-View Hierarchical Clustering (MHC). MHC essentially extends the FINCH Algorithm [20] in multi-view domain. Like FINCH does for single-view data, MHC recursively finds clusters in multi-view datasets in levels of granularity. Also, there is no hyperparameter selection. For the extension to multi-view framework, mainly two concepts are used: Cosine Distance Integration (CDI) and Nearest Neighbor Agglomeration (NNA). In the next subsection we take a deeper look at these two concepts.

Formulation

Cosine Distance Integration (CDI):
The main purpose of CDI is to compute an essential latent distance matrix from the individual distance matrices of each view. This latent matrix must contain the complementary information of all the views.

Following [3], let us assume H to be the latent representation of multiple views $\{X^{(i)}\}_{i=1}^{v}$, ie. H contains the complementary information of all the views. Given H, the data for each view can be reconstructed as follows:

$$X^{(i)} = P^{(i)}H, {P^{(i)}}^T P^{(i)} = I \tag{3.2}$$

where $P^{(i)}$ is an orthgonal matrix of the i-th view.

Distances between samples in each view are calculated using the cosine distance. More specifically, let a and b be two data samples represented as x_a and x_b respec-

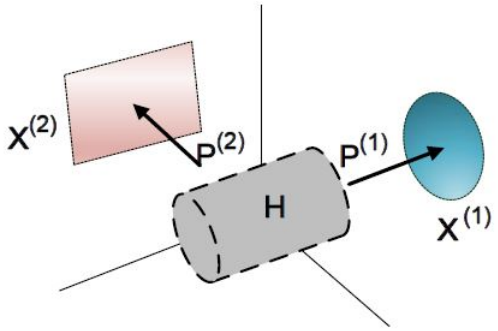


Figure 3.1: Illustration of how two views $X^{(1)}$ and $X^{(2)}$ are partially projected by $P^{(1)}$ and $P^{(1)}$ from the same underlying latent representation H (Image source: [3])

tively. The cosine distance between them in the i-th view is given by:

$$d_{ab}^{(i)} = 1 - \frac{{x_a^{(i)}}^T x_b^{(i)}}{\sqrt{{x_a^{(i)}}^T x_a^{(i)}}\sqrt{{x_b^{(i)}}^T x_b^{(i)}}} \tag{3.3}$$

On the other hand, in the latent representation the distance between the above two samples a and b can be computed as:

$$d_{ab}^{*} = 1 - \frac{h_a^T h_b}{\sqrt{h_a^T h_a}\sqrt{h_b^T h_b}} \tag{3.4}$$

where h_a and h_b are the representations of a and b respectively in the latent representation.

Now, putting $x_a^{(i)} = P^{(i)} h_a$ and $x_b^{(i)} = P^{(i)} h_b$ in the cosine distance formula of Equation (3.3), it can be shown that ${d_{ab}}^{(i)} = {d_{ab}}^{*}$ theoretically. As a result, in practice, the essential cosine distance between the two samples a and b can be given as

$$d_{ab}^{*} = \frac{1}{v}\sum_{i=1}^{v} d_{ab}^{(i)} \tag{3.5}$$

In the next step of NNA, this learned essential distance matrix is used for finding clusters in the data.

Nearest Neighbor Agglomeration (NNA):
NNA essentially applies FINCH algorithm by using the latent distance matrix computed during NNA. It works on the same assumption as FINCH that data samples and their first nearest neighbors should be clustered into the same partition [40][41]. This merging results in long chains of samples which are grouped together in a single iteration.

To be more specific, the essential matrix computed during CDI is used to generate a graph $\mathcal{G}$ in which indices corresponding to a data sample and its first nearest neighbor are connected.

$$\mathcal{G}(a,b) = \begin{cases} 1, \text{ if } a = nearest(b) \text{ or } b = nearest(a) \\ 0, \text{ otherwise} \end{cases} \tag{3.6}$$

where $nearest(i)$ gives the first nearest neighbor of data sample i.

Connected components of the graph $\mathcal{G}$ form the clusters. But this clustering being in the latent representation, they have to be transferred to the individual views. The idea is to consider the newly obtained clusters as individual data points for the CDI in the next iteration. For this, a new data sample corresponding to a cluster is obtained by averaging over all the data points belonging to that cluster. The nearest neighbors in this step can be obtained using fast approximation algorithms like k-d tree which help in reducing the time complexity of this algorithm to $\mathcal{O}(nlogn)$.

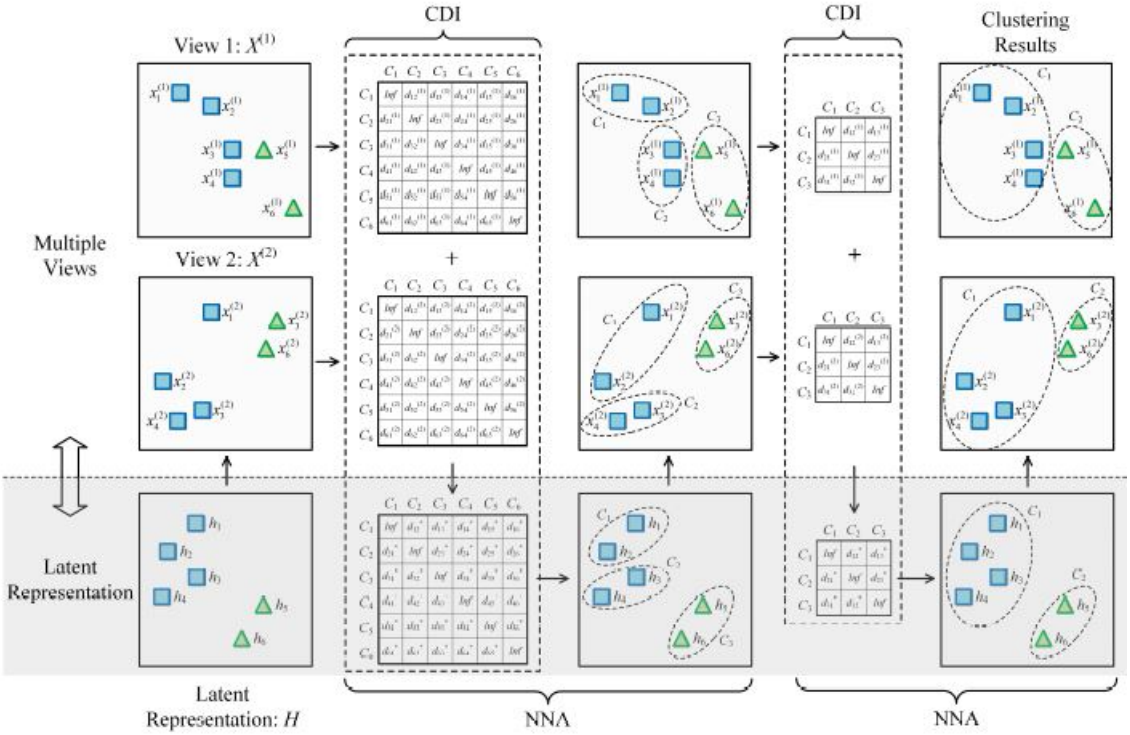


Figure 3.2: Illustration of CDI and NNA steps over successive iterations (Image source: [4])

The Algorithm

All data samples are first treated as individual clusters. Then the CDI and NNA steps are performed one after the other recursively, until we have only one cluster.

Algorithm 5 depicts the MHC Algorithm when a particular number of clusters is not required. Like in FINCH, results are obtained in different levels of granularity and the partitions obtained in each of these iterations have their own inherent meaning.

MHC can also be modified with Algorithm 6 to operate when a fixed number of clusters is required. It simply selects the clustering with closest number of partitions greater than the required number of clusters. From that clustering, two clusters are merged at a time following the approach of AGNES [42], which, contradictory to MHC is a single-view clustering technique.

Observations

MHC, as we have mentioned can be looked upon as an extension to FINCH in multi-view framework. As a result it carries most of the advantages of FINCH. MHC merges multiple clusters at a time leading to the final clustering result in very few iterations even for large multi-view datasets. Having the ability to provide clustering results even when no required number of clusters is pre-specified is an added advantage in its own right as finding clustering results without mentioning the actual number of clusters is a topic of interest among current researchers. Performance comparisons shown in [4] shows the efficiency of MHC when compared to other state-of-the-art multi-view clustering algorithms.

Now question is, can the multi-view framework developed for MHC be used to incorporate OT into multi-view hierarchical clustering? Analysing this possibility is the core component of subsequent chapters of this work.

Algorithm 5: MHC Algorithm

Input : $\{X^{(i)}\}_{i=1}^{v}$, the data matrices for v views
Output: Clustering results at multiple levels of granularity, $\mathcal{R} = \{R_0, R_1, ...\}$

Performing the CDI:
for *i=1 to v* **do**
  Calculating distance matrix $D^{(i)}$ using cosine distance;
end
Getting essential distance matrix $D^* = \frac{1}{v}\sum_{i=1}^{v} D^{(i)}$;
Performing the NNA:
Constructing $\mathcal{G}$ based on D^* using Equation (3.6);
Getting clustering results R_0 based on $\mathcal{G}$;
Transferring current results R_0 to multiple views;
Calculating new data samples as new input data by averaging data samples in the same cluster;
Performing the CDI and NNA recursively:
while *current results have more than two clusters* **do**
  Performing the CDI:
  for *i=1 to v* **do**
    Calculating distance matrix $D_k^{(i)}$ based on new input using cosine distance;
  end
  Getting essential distance matrix $D_k^* = \frac{1}{v}\sum_{i=1}^{v} D_k^{(i)}$;
  Performing the NNA:
  Constructing $\mathcal{G}_k$ based on D_k^* using Equation (3.6);
  Getting clustering results R_k based on $\mathcal{G}_k$;
  Transferring current results R_k to multiple views;
  Calculating new data samples as new input data by averaging data samples in the same cluster;
end

Algorithm 6: MHC Algorithm with a fixed number of clusters

Input : $\{X^{(i)}\}_{i=1}^{v}$, K and the closest division clustering results $R_{closest}$, which has $|R_{closest}|$ clusters

Output: Clustering result R_K with the required number of clusters $|R_K| = K$

Initializing:

Calculating new data samples of multiple views as new input data by averaging samples in the same cluster according to $R_{closest}$;

for $iters = |R_{closest}| - |R_K|$ **do**

  Performing the CDI:

  for *i=1 to v* **do**

    Calculating distance matrix $D_{iters}^{(i)}$ based on latest R using cosine distance;

  end

  Getting essential distance matrix $D_{iters}^{*} = \frac{1}{v}\sum_{i=1}^{v} D_{iters}^{(i)}$;

  Merging two closest clusters:

  Getting new clustering results by merging two closest clusters based on D_{iters}^{*};

end

Chapter 4

Proposed Multi-View Hierarchical Clustering Algorithm

4.1 Incorporating Optimal Transport into Hierarchical Clustering with First-Nearest Neighbors

In this section, we aim to incorporate Optimal Transport (OT) into the single-view FINCH Algorithm [20]. The strategy is quite similar to that done in HC-OT [23] while incorporating OT into the traditional hierarchical clustering framework.

4.1.1 Formulation

Our approach remains the same as that in FINCH; however the modification comes in the step where the first nearest neighbor of each cluster is calculated. The idea of FINCH is to use euclidean or cosine distance between each cluster in a stage and use these distances to find the first nearest neighbor of each cluster. In addition, for large datasets, fast approximation algorithms like k-d tree can be leveraged for finding these first nearest neighbors much faster.

Our approach uses OT to calculate the distances between the clusters. Then these OT distances are used to find the first nearest neighbors contrary to using simple euclidean or cosine distance measures. Obviously in doing so, we give up the ability to use k-d tree as in FINCH to approximate the nearest neighbors faster. However, our focus is mainly to improve the clustering performance here.

Defining the distance between two clusters using OT

The natural question to ponder is how the distance between two given clusters can be defined using OT. We follow the setup for the incorporation of OT suggested by [23]. As discussed in Section 3.1.1, given two clusters C_1 and C_2, the Sinkhorn distance between them, $S_\lambda(C_1, C_2)$, for $\lambda > 0$, is defined as follows. Let us assume that x_i is the i-th data point of C_1, $i = 1, ..., |C_1|$ and y_j is the j-th data point of C_2, $j = 1, ..., |C_2|$. Also, consider that μ and ν denote empirical distributions of the data points in the clusters C_1 and C_2 respectively. Thus $\mu = \sum_{i=1}^{|C_1|} \frac{1}{|C_1|} \delta_{x_i}$

and $\nu = \sum_{j=1}^{|C_2|} \frac{1}{|C_2|}\delta_{y_j}$. c_{ij} is the distance between x_i and y_j. Then we can say that $S_\lambda(C_1, C_2) = d_C^\lambda(\mu, \nu)$, the Sinkhorn distance between μ and ν.

Also note that this approximation is only valid if $|C_1|$ and $|C_2|$ are large enough. [23] follows that if both $|C_1|$ and $|C_2| > n/10$, then they can successfully use Sinkhorn distance, otherwise they revert to single/complete linkage to get the distance between the two clusters. We, on the other hand, have observed that other thresholds such as $n/15$, $n/20$ or $n/25$ may lead to much better performance on certain datasets for our algorithm and as a result it is worthwhile to treat this threshold as a hyper-parameter. Hence, we follow the rule that if both $|C_1|$ and $|C_2| > n/t$, where t can be chosen from a pre-defined set of values, then Sinkhorn distance can be successfully applied for approximating the distance between the two clusters. Otherwise, we use normal euclidean or cosine distance computed between the means of the two corresponding clusters as a measure of distance between them.

4.1.2 The Proposed FINCH-OT Algorithm

Algorithm 7 depicts the FINCH-OT Algorithm for clustering single-view data, when clustering of a specific given K number of clusters is not required. As can be seen, lines other than $9-10$ are similar to that of FINCH. In lines $9-10$, OT has been leveraged to compute the distance between the two clusters C_1 and C_2, if both $|C_1|$ and $|C_2| > n/t$. Otherwise $dist(C_1, C_2)$ calculates the distance between C_1 and C_2 using either traditional euclidean or cosine distance. The algorithm outputs valid partitions at different levels of granularity.

Algorithm 7: Proposed FINCH-OT Algorithm

Input : $X \in \mathbb{R}^{n\times d}$, λ, t: threshold
Output: Clustering results at multiple levels of granularity, $\mathcal{R} = \{R_0, R_1, ...\}$ where each R_i can be treated as a valid clustering of X

1 Calculate the distance matrix D_0 from X using euclidean/cosine distance;
2 Using D_0 find the *first nearest neighbor* of each sample;
3 Compute the nearest neighbor adjacency graph matrix $\mathcal{G}_0$ using Equation (3.1);
4 $R_0 =$ the *connected components* in $\mathcal{G}_0$;
5 $i = 1$;
6 **while** $|R_{i-1}| \geq 2$ **do**
7 Compute cluster means for each cluster in R_{i-1};
8 Prepare new data matrix $M \in \mathbb{R}^{|R_{i-1}|\times d}$ containing only these mean points as representatives for each cluster in R_{i-1};
9 Calculate D_i as follows:
10 $$d_i[C_1, C_2] = \begin{cases} S_\lambda(C_1, C_2), & \text{if } |C_1|, |C_2| > \frac{n}{t} \\ dist(C_1, C_2), & \text{otherwise} \end{cases};$$
11 Compute $\mathcal{G}_i$ from D_i using Equation (3.1);
12 $R_i =$ the *connected components* in $\mathcal{G}_i$;
13 $i = i + 1$;
14 **end**

Algorithm 8: Proposed FINCH-OT Algorithm for K clusters required

Input : $X \in \mathbb{R}^{n \times d}$; λ; t: threshold; K; $R_{closest}$ obtained from Algorithm 7 *st* $|R_{closest}| \geq K$, $|R_{closest+1}| < K$

Output: Partitioning R_K with $|R_K| = K$ number of clusters

$iters = |R_{closest}| - |R_K|$;
$R' = R_{closest}$;
while $iters \geq 1$ **do**
    Compute cluster means for each cluster in R'.
    Prepare new data matrix $M \in \mathbb{R}^{|R'| \times d}$ containing only these mean points as representatives for each cluster in R';
    Calculate D as follows:

$$d[C_1, C_2] = \begin{cases} S_\lambda(C_1, C_2), & \text{if } |C_1|, |C_2| > \frac{n}{t} \\ dist(C_1, C_2), & \text{otherwise} \end{cases};$$

    Compute the nearest neighbor adjacency graph matrix $\mathcal{G}$ from D using Equation (3.1);
    Find d_{min}, the minimum distance between all cluster pairs (k, l) for which $\mathcal{G}(k, l) = 1$;
    Keep the symmetric link in $\mathcal{G}$ for the cluster pair (a, b) corresponding to d_{min}, and set the rest to zero;
    Merge corresponding (a, b) clusters to get new R';
    $iters = iters - 1$;
end

As is the case with FINCH, our proposed FINCH-OT algorithm works if the required number of clusters K is provided as well. First, Algorithm 7 is run and the partition $R_{closest}$ which contains the nearest number of clusters greater than K is passed to Algorithm 8. The idea is to merge only the two nearest clusters in each step so that from $|R_{closest}|$ clusters we can come down to K clusters in $|R_{closest}| - K$ iterations. Similar to Algorithm 7, Sinkhorn distance is used to discern the distance between two clusters.

4.2 Extension to Multi-View Framework

Section 4.1 introduces the FINCH-OT Algorithm for single-view data. In this section, we look to build a multi-view hierarchical clustering scheme which can be looked upon as a multi-view extension of FINCH-OT.

4.2.1 Formulation

We build on the concepts introduced in MHC [4] which uses *Cosine Distance Integration (CDI)* and *Nearest Neighbor Agglomeration (NNA)* to form valid cluster partitions in levels of granularity as in FINCH-OT. Essentially, MHC only uses cosine distance as a measure between two clusters. This is mainly due to the assumption that if several views are represented by a Latent Representation H and $P^{(i)}$ is an orthogonal matrix of the i-th view, then

$$X^{(i)} = P^{(i)}H, {P^{(i)}}^T P^{(i)} = I \tag{4.1}$$

Now using the formula of cosine distance, [4] shows that using Equation (4.1) we will get $d_{ab}^{(i)} = d_{ab}^{*}$ theoretically, for two data samples a and b. Thus is practice, giving equal weight to each view, the latent distance between these two samples can be calculated as

$$d_{ab}^{*} = \frac{1}{v}\sum_{i=1}^{v} d_{ab}^{(i)} \tag{4.2}$$

Hence, using cosine distances allows us to compute a latent distance for all views between points or clusters by a simple average, while satisfying Equation (4.1).

Introducing Euclidean Distance Integration (EDI)

Experiments with FINCH-OT performed by us have suggested that euclidean distance may provide better results compared to cosine distance for some datasets. So it should be advantageous to incorporate euclidean distance in the multi-view setup as well.

Lemma 4.2.1. *Just as cosine distance, euclidean distance too satisfies $d_{ab}^{(i)} = d_{ab}^{*}$, and as a result can be fit into the existing MHC framework.*

Proof. Let $d_{ab}^{(i)}$ be the distance between samples a and b in the i-th view, and d_{ab}^{*} be the distance between them in the latent representation. Also let h_a and h_b represent samples a and b respectively in the latent representation. Now consider the square of the euclidean distance between samples a and b represented as $x_a^{(i)}$ and $x_b^{(i)}$ respectively in the i-th view.

$$\begin{aligned}
{d_{ab}^{(i)}}^2 &= ||x_a^{(i)} - x_b^{(i)}||_2^2 \\
&= ||P^{(i)}h_a - P^{(i)}h_b||_2^2 \quad \text{[Using Equation (4.1)]} \\
&= (P^{(i)}h_a - P^{(i)}h_b)^T(P^{(i)}h_a - P^{(i)}h_b) \\
&= \{(P^{(i)}h_a)^T - (P^{(i)}h_b)^T\}(P^{(i)}h_a - P^{(i)}h_b) \\
&= (P^{(i)}h_a)^T P^{(i)}h_a - (P^{(i)}h_a)^T P^{(i)}h_b - (P^{(i)}h_b)^T P^{(i)}h_a + (P^{(i)}h_b)^T P^{(i)}h_b \\
&= h_a^T {P^{(i)}}^T P^{(i)}h_a - h_a^T {P^{(i)}}^T P^{(i)}h_b - h_b^T {P^{(i)}}^T P^{(i)}h_a + h_b^T {P^{(i)}}^T P^{(i)}h_b \\
&= h_a^T h_a - h_a^T h_b - h_b^T h_a + h_b^T h_b \\
&= (h_a - h_b)^T(h_a - h_b) \\
&= ||h_a - h_b||_2^2 \\
&= {d_{ab}^{*}}^2
\end{aligned}$$

Since, distance cannot be negative, $d_{ab}^{(i)} = d_{ab}^{*}$.

So, euclidean distance can be used as well in the MHC setting in place of cosine distance if required. In that case, we can have an *Euclidean Distance Integration (EDI)* step to replace the *Cosine Distance Integration (CDI)* step. □

Incorporating Optimal Transport

Optimal Transport is used in a similar manner to that in FINCH-OT. The *Cosine/Euclidean Distance Integration* step now consists of using OT to calculate $d_{ab}^{(i)}$,

the distances between cluster a and b in each view, if the corresponding clusters satisfy the desired criteria of having enough data points. These distances of each individual view is then averaged to come up with d^*_{ab}, the latent distance matrix between a and b using Equation (4.2).

4.2.2 The Proposed MVHC-OT Algorithm

We call the multi-view version of our algorithm as Multi-View Hierarchical Clustering with Optimal Transport (MVHC-OT). Algorithm 9 depicts the MVHC-OT Algorithm when no required number of clusters is explicitly mentioned. Here we provide a brief outline of the Algorithm. Initially we start with each data sample as individual clusters. Then we have the CDI or EDI step as required for the corresponding dataset. Cosine or Euclidean distances are computed among data points in each view. Following this, the essential distance matrix D^*_0 is computed by averaging the distance matrices of all the views using Eqaution (4.2). We then move on to the NNA step, where the first nearest adjacency graph $\mathcal{G}_0$ is computed from D^*_0 using Equation (3.6). Connected components of $\mathcal{G}_0$ provide the clustering result at the first level R_0. We now have the clusters in the latent representation. These partitions are then transferred to the individual views.

The idea after this is to perform CDI/EDI and NNA recursively until all data points belong to the same cluster. However, the distances in each view are now calculated using OT. To be more specific, we check if the required conditions for applying OT as a distance measure between two clusters are satisfied or not. If so, we apply Sinkhorn distance to calculate the inter-cluster distance. Else, we calculate the distance between their means using traditional cosine/euclidean distance as suggested in MHC.

One thing to note is that we do not use OT in the first CDI/EDI step. To be more general, one can keep the same conditions for applying OT in this step too. However, observe that in this step each cluster is actually one data point, ie. $|C_j| = 1 \ \forall j = 1, ..., n$. Thus according to the conditions for applying OT, we must have $n/t < 1$ if we are to apply OT. As we will see in Chapter 6, t is chosen from the set $\{10, 15, 20, 25\}$. Hence, if $n = 25$ or lower, then only this condition may be satisfied, which is unlikely in most cases. Also, it can be very intuitively understood that with only one data point directly computing euclidean or cosine distance serves the purpose.

In case a partitioning with exactly K clusters are required, at first, Algorithm 9 is run and the partition $R_{closest}$ which contains the nearest number of clusters greater than K is passed to Algorithm 10. The idea is to merge only the nearest clusters in each step so that from $|R_{closest}|$ clusters we can come down to K clusters in $|R_{closest}| - K$ iterations. Like before, distances are once again calculated using OT only if the required conditions are satisfied.

Algorithm 9: Proposed MVHC-OT Algorithm

Input : $\{X^{(i)}\}_{i=1}^{v}$: data matrices for v views, λ
Output: Clustering results at multiple levels of granularity, $\mathcal{R} = \{R_0, R_1, ...\}$

Perform CDI/EDI:
for *i=1 to v* **do**
  calculate $D_0^{(i)}$ from $X^{(i)}$ using suitable distance measure;
end
Calculate latent distance matrix D_0^* using Equation (4.2);
Perform NNA:
Calculate *first nearest neighbor graph* $\mathcal{G}_0$ based on D_0^* using Equation (3.6);
$R_0 =$ the *connected components* of $\mathcal{G}_0$;
Transfer clustering results to each view;
$k = 1$;
Perform CDI/EDI and NNA recursively:
while $|R_{k-1}| \geq 2$ **do**
  for *i=1 to v* **do**
    calculate $D_k^{(i)}$ as follows:

$$d_k^{(i)}[C_1, C_2] = \begin{cases} S_\lambda(C_1, C_2), & \text{if } |C_1|, |C_2| > \frac{n}{t} \\ dist(mean \text{ of } C_1, mean \text{ of } C_2), & \text{otherwise} \end{cases};$$

  end
  Get D_k^* using Equation (4.2);
  Form $\mathcal{G}_k$ based on D_k^* using Equation (3.6);
  $R_k =$ the *connected components* of $\mathcal{G}_k$;
  Transfer clustering results to each view;
  $k = k + 1$;
end

Algorithm 10: Proposed MVHC-OT Algorithm for K clusters required

Input : $\{X^{(i)}\}_{i=1}^{v}$; λ; t: threshold; K; $R_{closest}$ obtained from Algorithm 9 *st* $|R_{closest}| \geq K$, $|R_{closest+1}| < K$

Output: Partitioning R_K with $|R_K| = K$ number of clusters

$iters = |R_{closest}| - |R_K|$;
$R' = R_{closest}$;
while $iters \geq 1$ **do**
  for *i=1 to v* **do**
    calculate $D^{(i)}$ as follows:

$$d^{(i)}[C_1, C_2] = \begin{cases} S_\lambda(C_1, C_2), & \text{if } |C_1|, |C_2| > \frac{n}{t} \\ dist(mean \text{ of } C_1, mean \text{ of } C_2), & \text{otherwise} \end{cases};$$

  end
  Get latent distance matrix D^*;
  Form *first nearest neighbor graph* $\mathcal{G}$ based on D^*;
  Find d_{min}, the minimum distance between all cluster pairs (k, l) for which $\mathcal{G}(k, l) = 1$;
  Keep the symmetric link in $\mathcal{G}$ for the cluster pair (a, b) corresponding to d_{min}, and set the rest to zero;
  Merge corresponding (a, b) clusters to get new R';
  $iters = iters - 1$;
end

FINCH-OT as a Special Case of MVHC-OT

If we carefully observe Algorithms 7 and 8 for FINCH-OT and Algorithms 9 and 10 for MVHC-OT, we can see that FINCH-OT, in reality, is just a special case of MVHC-OT for single-view data. This is because in MVHC-OT, the distance matrices of each view are simply averaged to calculate the latent distance matrix. In case of only one view (as in the case of FINCH-OT), the distance matrix of the only view will be same as the latent distance matrix. Hence, MVHC-OT suffices for single-view data as well.

Chapter 5

Performance Analysis of Proposed MVHC-OT Algorithm

5.1 Complexity Analysis

The time complexity of MVHC-OT is understandably going to be higher than that of MHC due to the incorporation of OT during the computation of the distances between two clusters which takes $\mathcal{O}(mn)$ time when the two clusters have m and n points respectively. Here we analyse in detail the time complexity of our MVHC-OT Algorithm.

The building of the initial distance matrices for v views takes $\mathcal{O}(vn^2)$ time. This initial distance matrix may be considered as pre-computed as well, as is done in many previous works. Note that since every data point is a cluster in its own here, that is, every cluster has only one data point, the distances can never be computed using OT in a practical scenario. Then, the computation of the latent distance matrix D_0^* takes $\mathcal{O}(vn^2)$. Now, finding the closest neighbor of each point and computing the nearest neighbor adjacency matrix from the nearest neighbor graph takes $\mathcal{O}(n^2)$ time. From this matrix, the clusters for the next stage can be derived by finding the connected components in $\mathcal{O}(n)$ time. Say, we find l_1 connected components (l_1 has no direct relationship with n, other than the fact that $l_1 << n$). Hence, iteration 1 takes $\mathcal{O}(n^2)$ time in total.

In the next iteration, the representative point for each cluster of iteration 1 is computed by averaging all the points in the corresponding cluster which takes $\mathcal{O}(vn)$ in total. Now, during the computation of D_1, the distance matrix for view 1, distances may be computed using OT if the corresponding clusters satisfy the required criteria. So, in the worst case, for each of the l_1^2 positions of the matrix D_1 we may need to compute distances using OT. This, for v views, requires $\mathcal{O}(vl_1^2n^2)$ time in total. Then, the computation of the latent distance matrix D_1^* takes $\mathcal{O}(vl_1^2)$. Now, finding the closest neighbor of each point and computing the nearest neighbor adjacency matrix from the nearest neighbor graph takes $\mathcal{O}(l_1^2)$ time. From this matrix, the clusters for the next stage can be derived by finding the connected components in $\mathcal{O}(l_1)$ time. Say, we find l_2 connected components (l_2 has no direct relationship with n or l_1, other than the fact that $l_2 << l_1 << n$). Hence, iteration 2 takes $\mathcal{O}(n^2)$ time with respect to n in total as well. Similarly, the next stages of the MVHC-OT algorithm takes $\mathcal{O}(n^2)$ time as well due to the distance matrix calculation step which uses OT (which takes $\mathcal{O}(n^2)$) when certain conditions are

satisfied.

Note that we have only a limited number of iterations, say k. So, overall our proposed MVHC-OT algorithm (Algorithm 9) costs $\mathcal{O}(kn^2)$ time or $\mathcal{O}(n^2)$ time in n.

The above analysis holds for the case when the algorithm is not asked to output partitions with a specific number of clusters. In case it is asked to do so, we need to use Algorithm 10. Algorithm 10 essentially runs Algorithm 9 first, which takes $\mathcal{O}(n^2)$ as we have derived earlier. Then starting from the closest partitioning level larger than the required number of clusters, it merges only two clusters at a time using OT distances when conditions are satisfied. So suppose we start at partitioning with number of clusters l_a and need to reach number of clusters K. So we will have $l_a - K$ iterations and each will take $\mathcal{O}(n^2)$ in the worst case. Now unless, $l_a = n$, the number of data samples, we will still end up with a worst case total time complexity of $\mathcal{O}(n^2)$. If $l_a = n$, we shall get a total time complexity of $\mathcal{O}(n^3)$. However, as we shall see in Section 6.2, in none of our experiments for single-view or multi-view data, have we faced a scenario where $l_a = n$.

5.2 Experimental Evaluations

In this section, the performance evaluation of our MVHC-OT Algorithm on both Single-View and Multi-View datasets has been performed. Our algorithm has been implemented in Python 3.7 and the results obtained have been performed on a Windows machine with a 4-core 2.30 GHz processor and 8 GB RAM.

5.2.1 Datasets

Single-View Data

8 real-world single-view datasets have been used for evaluating and comparing our proposed MVHC-OT technique with state-of-the-art algorithms. These datasets are as follows: Leukemia [43], Iris, Glass, Wine, Zoo, Seeds, Brain [44] and SRBCT [45]. Iris, Glass, Wine, Zoo and Seeds datasets have been collected from the Keel [46] and UCI machine learning [47] repositories. Influential Feature PCA (IF-HCT-PCA) [48] is used for pre-processing the Brain and SRBCT datasets. A general description of these datasets have been provided in Table 5.1.

Multi-View Data

In the multi-view scenario, for the purpose of evaluation and comparison with state-of-the-art algorithms, we use 5 multi-view datasets: Caltech101 [49], UCI [50], Football [51], Olympics [51], Politicsie [51] and 100 Leaves [52]. A general description of these datasets have been provided in Table 5.2.

5.2.2 Compared Algorithms

For Single-View Data

In our study, we compare the performance of our MVHC-OT approach with several existing state-of-the-art single-view hierarchical clustering algorithms like Single

Dataset	**Samples**	**Features**	K
Leukemia	72	3571	2
Iris	150	4	3
Glass	214	9	6
Wine	178	13	3
Zoo	101	16	7
Brain	42	5597	5
SRBCT	63	2308	4
Seeds	210	7	3

Table 5.1: Description of real-world single-view datatsets

Dataset	**Samples**	**Views**	**Number of features in each view**	K
Caltech101	9144	6	$\{48, 40, 254, 512, 928, 1984\}$	102
UCI	2000	3	$\{240, 76, 6\}$	10
Football	248	9	$\{248, 248, 3601, 7814, 248, 248, 248, 248, 11806\}$	20
Olympics	464	9	$\{464, 464, 3097, 4942, 464, 464, 464, 464, 18455\}$	28
Politicsie	348	9	$\{348, 348, 1051, 1047, 348, 348, 348, 348, 14377\}$	7
100Leaves	1600	3	$\{64, 64, 64\}$	100

Table 5.2: Description of real-world multi-view datatsets

Linkage [53][54], Complete Linkage [55], UPGMA or Average Linkage (Unweighted Pair Group Method with Arithmetic Mean) [56], WPGMA (Weighted Pair Group Method with Arithmetic Mean) [57], UPGMC (Unweighted Pair Group Method with Arithmetic Centroid) [58], WPGMC (Weighted Pair Group Method with Centroid) [58], SPHC (with average linkage) [19], FINCH (First Integer Neighbor indices to produce a Cluster Hierarchy) [20] and HC-OT (Hierarchical Clustering with Optimal Transport) [23]. Results for FINCH are obtained by running the code made available publicly by the authors. For the rest of the algorithms, results are taken from the very recently published [23]. Note that among the above mentioned algorithms only FINCH can give a final clustering result even without knowing the actual number of clusters in the input dataset.

For Multi-View Data

For the multi-view realm, we evaluate and compare the performance of our MVHC-OT Algorithm with several state-of-the-art multi-view clustering algorithms such as k-means, Low Rank Representation (LRR) [59], Auto-weighted Multiple Graph Learning (AMGL) [10], Latent Multi-View Subspace Clustering (LMSC) [3], Binary Multi-View Clustering (BMVC) [11], Graph-based Multi-View Clustering (GMC) [9], Multi-view Clustering without Parameter Selection (COMIC) [12], Large Scale Multi-View Subspace Clustering in Linear Time (LMVSC) [7] and Multi-View Hierarchical Clustering (MHC) [4]. For MHC we reproduce the code following the

algorithm in [4]. Codes provided by the authors are used with optimal parameter settings to obtain results for BMVC, GMC, COMIC and LMVSC. In case of k-means, the function available in scikit-learn [60] is used. For the remaining algorithms, we take the results as reported by [4]. Among the aforementioned algorithms, only MHC possesses the capability of providing multiple valid partitions in levels of granularity even when no specific K is mentioned as input.

5.2.3 Evaluation Metrics

For Single-View Data

For single-view datasets we use the Normalized Mutual Information (NMI) score to evaluate the performance of our proposed algorithm and to compare it with other state-of-the-art algortihms.

For Multi-View Data

For multi-view datasets we use the Normalized Mutual Information (NMI) score and Accuracy as means to compare the performance of our proposed algorithm with state-of-the-art multi-view clustering algortihms.

5.2.4 Parameter Settings

The MVHC-OT algorithm has three parameters: the distance measure (euclidean/ cosine) to be used in the cost matrix of OT; λ, the regularization term for OT and the threshold t, such that the distance between two clusters can be computed by Sinkhorn distance if both contain more than n/t data samples.

Our experimental observations show that for almost all the datasets (both single and multi-view), the value of λ can actually be set to anything from say 1.0 to 1000.0 without really changing the results. However for a couple of datasets, solving the entropic regularized form of OT (Equation 2.4) throws an error while using the *POT: Python OT* package [61] with $\lambda = 1.0$. So sticking to $\lambda = 1000.0$ serves our purpose for all the datasets as other valid values of λ do not effect the performance. Hence effectively, our true number of parameters becomes two instead of three.

Next, for setting t, we run our algorithm for each value from the set $\{10, 15, 20, 25\}$ and select the one which gives best performance. Similarly for the distance measure, either euclidean or cosine is chosen. To be more specific, t and the distance measure may be chosen by grid search method.

For Single-View Data

For the case **when the required number of clusters is not pre-specified**, the hyperparameter settings under which we get the results obtained for MVHC-OT (Table 5.3) are as follows (λ as mentioned earlier is always set to 1000.0). Euclidean distance is used in case of Glass, Zoo, SRBCT and Seeds datasets, while cosine distance is used for the rest. $t = 10$ is used for Wine, Zoo and Brain datasets. $t = 15$ is used for Iris,Glass and Seeds, while $t = 25$ is used for Leukemia and SRBCT. Note that it may be the case that other valid values of t or distance measure may also give equally good performance for a dataset. The values mentioned here are simply the ones that give the best performance and were chosen first during grid search.

In contrast, **when the required number of clusters is pre-specified**, we get the results obtained for MVHC-OT (Table 5.5) under the following parameter settings. Euclidean distance is used in case of Zoo, SRBCT and Seeds datasets, while Cosine distance is used for the rest. $t = 10$ is used for Wine, Zoo, Brain and Seeds datasets. $t = 15$ is used for Iris and Leukemia, while $t = 20$ is used for Glass, and $t = 25$ for SRBCT.

For Multi-View Data

When the required number of clusters is not pre-specified, the hyperparameter settings under which we get the results obtained for MVHC-OT (Table 5.4) are as follows. Euclidean Distance is used for UCI and 100Leaves datasets. Cosine distance is used for the rest. Caltech101, UCI, 100Leaves, Olympics and Politicsie datasets all use $t = 10$. $t = 20$ is used for Football dataset.

On the contrary, **when the required number of clusters is pre-specified**, we get the results obtained for MVHC-OT (Table 5.6 and Table 5.7) under the following parameter settings. Euclidean distance is used for only the 100Leaves dataset. For all the other datasets cosine distance is used. t is set to 10 for Caltech101, Politicsie and 100Leaves datasets. For UCI and Olympics t is chosen to be equal to 15 and 20 respectively. Football dataset on the other hand uses $t = 25$.

5.2.5 Performance when Pre-specified K number of clusters not required

For Single-View Datasets

Apart from our MVHC-OT algorithm, only FINCH has the capability of providing clustering results when K is not provided beforehand among our compared algorithms. Table 5.3 shows the comparative results between the two algorithms on the 8 real-world datasets mentioned previously. Underlined partitioning levels mean that one of the levels correctly identifies the exact number of clusters.

The Table shows the number of clusters identified at each level of clustering by the two algorithms. As we can see it takes just about 2-4 steps for both the algorithms to find the clusterings. This is primarily because the number of clusters in the first level is way less than the number of samples in all the datasets. And in each further level it keeps reducing at a rapid rate. In addition, MVHC-OT accurately finds the actual required number of clusters in one of its levels for four out of the eight datasets, while FINCH does so in only one.

The NMI index is computed by using the clustering at the level where the number of clusters is closest to the actual number of clusters K. We can see that the addition of OT to compute the distances in MVHC-OT definitely gives a boost in the performance as for all the 8 datasets MVHC-OT performs either as good as, or better than FINCH.

For Multi-View Datasets

In this Section we take a look at the multi-view scenario, which successfully outputs a set of partitions in levels of granularity just like its single-view counterpart. Each of these partitions can be intuitively thought as a valid clustering. In fact, Table

Dataset	Actual K	MVHC-OT			FINCH		
		NMI	Closest Level	Levels	NMI	Closest Level	Levels
Leukemia	2	**0.774**	2	$\underline{\{17, 2\}}$	0.177	3	$\{17, 3, 1\}$
Iris	3	**0.8705**	3	$\underline{\{38, 12, 3, 1\}}$	**0.8705**	3	$\underline{\{38, 12, 3, 1\}}$
Glass	6	**0.345**	4	$\{52, 13, 4, 1\}$	0.321	4	$\{52, 13, 4, 1\}$
Wine	3	**0.449**	3	$\underline{\{47, 12, 3, 1\}}$	0.381	4	$\{54, 14, 4, 1\}$
Zoo	7	**0.788**	5	$\{25, 5, 1\}$	**0.788**	5	$\{25, 5, 1\}$
Brain	5	**0.624**	8	$\{8, 2\}$	**0.624**	8	$\{8, 2\}$
SRBCT	4	**0.646**	4	$\underline{\{14, 4, 1\}}$	0.598	5	$\{14, 5, 1\}$
Seeds	3	**0.593**	3	$\underline{\{62, 13, 3, 1\}}$	0.542	4	$\{62, 13, 4, 1\}$

Table 5.3: Comparative results between MVHC-OT and MVHC on real-world single-view datasets when K is not known.

5.4 shows that the closest number of clusters in a partition found by MVHC-OT is indeed very close to K, the true number of clusters.

In case of multi-view data, apart from MVHC-OT only MHC is known to produce similar partitions in levels of granularity even if the actual number of clusters in the data is not provided explicitly. Table 5.4 compares the partition levels obtained by MVHC-OT and MHC, and also depicts the performances in terms of NMI index for the partition containing the closest number of clusters to K. We can see that MVHC-OT performs better or at least as good as MHC on all the datasets.

Dataset	Actual K	MVHC-OT			MHC		
		NMI	Closest Level	Levels	NMI	Closest Level	Levels
Caltech101	102	**0.448**	132	$\{1288, 132, 19, 7, 2\}$	0.444	133	$\{1173, 133, 18, 5, 1\}$
UCI	10	**0.903**	8	$\{413, 80, 19, 8, 3, 1\}$	0.824	9	$\{396, 77, 23, 9, 4, 1\}$
Football	20	**0.824**	16	$\{44, 16, 3, 1\}$	0.821	16	$\{44, 16, 3, 1\}$
Olympics	28	**0.905**	25	$\{86, 25, 5, 1\}$	**0.905**	25	$\{86, 25, 5, 1\}$
Politicsie	7	**0.773**	6	$\{49, 6, 2\}$	**0.773**	6	$\{49, 6, 1\}$
100 Leaves	100	**0.977**	106	$\{371, 106, 26, 4, 1\}$	0.956	115	$\{384, 115, 27, 6, 1\}$

Table 5.4: Comparative results between MVHC-OT and MHC on real-world multi-view datasets when K is not known

5.2.6 Performance when Pre-specified K number of clusters required

For Single-View Datasets

Table 5.5 shows the comparative results between MVHC-OT and the state-of-the-art hierarchical methods in terms of the NMI index on the 8 real-world datasets mentioned previously when a particular K number of clusters are required. As can be observed, MVHC-OT gives best performance on 5 of the 8 datasets, and is evidently much more consistent than the rest of the algorithms.

Specifically we would like to make a note of the performance of MVHC-OT for *high dimensional datasets*. A dataset is called high dimensional when the number of features exceeds the number of samples. Referring back to Table 5.1 we can see that according to our definition, Leukemia, Brain and SRBCT are high dimensional datasets. Observe the performances of MVHC-OT and FINCH for these datasets

Method	Leukemia	Iris	Glass	Wine	Zoo	Brain	SRBCT	Seeds
MVHC-OT	**0.903**	**0.8705**	**0.416**	0.449	0.784	**0.694**	**0.646**	0.593
FINCH	0.124	**0.8705**	0.379	0.449	0.788	0.587	0.540	0.593
HC-OT	0.9023	**0.8705**	0.2737	0.5725	**0.8637**	0.4997	0.4367	**0.701**
Single	0.0171	0.7175	0.0882	0.0348	0.6594	0.1972	0.1039	0.0198
Complete	0.6511	0.7221	0.1556	**0.6143**	0.8429	0.4158	0.3683	**0.701**
UPGMA	0.0034	0.8057	0.1126	0.01844	0.7512	0.3777	0.4184	0.6491
WPGMA	0.3817	0.7979	0.1024	0.5582	0.7781	0.3531	0.4184	0.5373
WPGMC	0.0171	0.7777	0.0724	0.0168	0.1843	0.3197	0.1007	0.4019
UPGMC	0.0411	0.7175	0.0833	0.018	0.1805	0.2299	0.1039	0.5764
SPHC	0.7740	0.7593	0.3910	0.4315	0.6134	0.4602	0.1282	0.6199

Table 5.5: Performance comparison in terms of NMI index of MVHC-OT with state-of-the-art hierarchical clustering algorithms on real-world single-view datatsets when K is known

in Table 5.5. Note that MVHC-OT for a single-view dataset is essentially just the incorporation of OT into FINCH. As suggested by Table 5.5, simply incorporating OT improves the performance by a large margin in case of all the three aforementioned datasets. While for the remaining datasets, we can see that performance either improves slightly or remains the same. The poor performance of FINCH may be attributed to the curse of dimensionality phenomenon [62] often observed in high dimensional datasets. The massive improvement in performance for MVHC-OT is suggestive that OT is more resistant to the effects of curse of dimensionality. If indeed so, OT may be highly effective in certain other applications which operate on high dimensional data.

For Multi-View Datasets

Table 5.6 shows the comparative analysis of MVHC-OT with the state-of-the-art multi-view clustering techniques with respect to the NMI index. The values in bold denote the best performance for a particular dataset, while the underlined values denote the second best performance for that particular dataset. We can clearly see that in case of majority of the datasets, MVHC-OT consistently gives either the best or second-best performance.

The performance comparison in terms of Accuracy is shown in Table 5.7. It can be observed that for all the datasets MVHC-OT consistently performs either the best or second-best.

Once again like in case of single-view datasets, we would like to specifically investigate the performance of MVHC-OT on *high-dimensional datasets*. We have seen that for single-view datasets, incorporation of OT massively boosts performance. Does the same hold true for multi-view datasets as well? According to Table 5.2, Foootball, Olympics and Politicsie datasets can be considered to be high dimensional as they have at least one view in which the number of features exceeds the number of samples. Now let us compare the performances of MHC and our MVHC-

OT algorithm (which is essentially MHC with the incorporation of OT and EDI for some datasets) for these datasets from Table 5.6 and Table 5.7. For both NMI and Accuracy we do see an improvement in performance in all the cases, especially for the Politicsie dataset. However, here the effect of OT for specifically high dimensional datasets is not as conclusive as in the single-view case. This is because even for non-high dimensional datasets, MVHC-OT is often seen to give a similar large improvement in performance as well.

Method	**Caltech101**	**UCI**	**Football**	**Olympics**	**Politicsie**	**100 Leaves**
MVHC-OT	0.449	**0.933**	0.875	**0.933**	0.741	**0.981**
MHC ('20)	0.449	0.894	0.862	0.891	0.332	0.949
LMVSC (AAAI '20)	0.263	0.712	0.681	0.705	0.474	0.785
COMIC (ICML '19)	0.288	0.892	0.797	0.832	0.720	0.700
GMC (TKDE '19)	0.345	0.812	**0.879**	0.875	0.753	0.929
BMVC (TPAMI '18)	**0.505**	0.796	0.749	0.807	0.636	0.909
LMSC (CVPR '17)	0.485	0.782	0.84	0.891	0.684	0.877
AMGL (IJCAI '16)	0.391	0.798	0.802	0.810	0.764	0.890
LRR (TPAMI '13)	0.495	0.768	0.85	0.867	**0.779**	0.736
k-means	0.495	0.75	0.567	0.475	0.519	0.838

Table 5.6: Performance comparison in terms of NMI index of MVHC-OT with state-of-the-art multi-view clustering algorithms on real-world multi-view datatsets

Method	**Caltech101**	**UCI**	**Football**	**Olympics**	**Politicsie**	**100 Leaves**
MVHC-OT	0.286	**0.969**	0.859	**0.869**	0.836	**0.94**
MHC ('20)	0.286	0.83	0.806	0.791	0.491	0.828
LMVSC (AAAI '20)	0.112	0.714	0.657	0.631	0.569	0.561
COMIC (ICML '19)	0.111	0.940	0.772	0.759	0.713	0.407
GMC (TKDE '19)	0.195	0.733	**0.883**	0.819	**0.856**	0.824
BMVC (TPAMI '18)	**0.288**	0.783	0.685	0.724	0.721	0.776
LMSC (CVPR '17)	0.251	0.859	0.795	0.804	0.690	0.748
AMGL (IJCAI '16)	0.238	0.764	0.744	0.689	0.816	0.727
LRR (TPAMI '13)	0.241	0.871	0.834	0.780	0.681	0.462
k-means	0.232	0.762	0.464	0.38	0.598	0.641

Table 5.7: Performance comparison in terms of Accuracy of MVHC-OT with state-of-the-art multi-view clustering algorithms on real-world multi-view datatsets

5.2.7 Sensitivity Analysis

We effectively use two parameters in our model: the threshold t and the distance measure euclidean or cosine. Hence it is interesting to investigate how the performance of MVHC-OT varies as we tweak these two parameters. We provide a detailed Sensitivity Analysis in this Section. Note that t can take values from the set $\{10, 15, 20, 25\}$.

For Single-View Datasets

Figure 5.1 shows the sensitivity analysis of NMI with respect to threshold t and distance measure (euclidean/cosine) for the single-view datasets used in our experiment. We also consider the NMI index value for FINCH with euclidean and cosine distances as baselines in the plots.

First, let us try to analyse in terms of the distance measure. The plots suggest that in a lot of the cases euclidean distance does actually give better performance than cosine distance. And both the distance measures give similar results in a couple of the datasets (Leukemia, Brain). Exploiting these cases is the major reason to treat the distance measure as a parameter in our model.

Probably the most significant parameter of our model is the threshold t. Stability of the performance with change in the parameter is a very important thing in any machine learning model; we never want our outputs to vary too much with the parameters, especially without any trend. If we take a look at the plots obtained, in quite a lot of the cases, both for euclidean and cosine distances, we find that for all the values of t, the NMI index remains constant. In the other cases, we see that for smaller values of t the NMI value is on the lower side (close to the baseline case of FINCH with the corresponding distance measure). And then as t is increased we reach the optimal performance along with some fluctuations in some cases (Leukemia, SRBCT).

However, the most important thing to note is that whatever be the value of t, the performance almost always stays same as or better than the corresponding FINCH baselines with the same distance measures. Only in case of SRBCT the NMI value drops well below the baselines for a while before improving.

Another point to observe is that at least for the datasets we have experimented on, choosing t from a smaller set of $\{15, 25\}$ would still give us same best performances.

For Multi-View Datasets

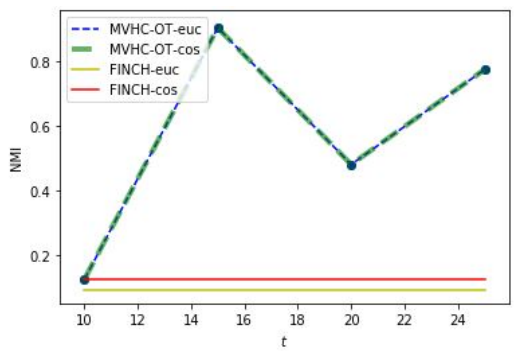
Figure 5.2 and Figure 5.3 show the sensitivity analysis of NMI and Accuracy respectively with respect to threshold t and distance measure (euclidean/cosine) for the multi-view datasets used in our experiment. The performance for MHC is also plotted as a baseline.

As far as the distance measure is concerned, we observe that unlike the case for single-view datasets, euclidean distance does not provide much improvement in performances. Only for the 100Leaves dataset, euclidean distance gives better performance than cosine. This however motivates us to still incorporate the distance measure as a parameter in our model as it may be beneficial for some other datasets which we have not used in our experiments.

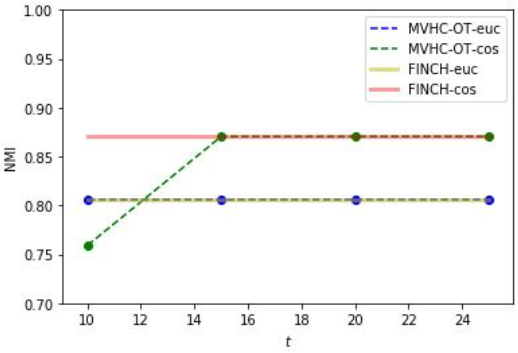
With respect to different values of t, similar to the case for single-view datasets, MVHC-OT shows good stability in performance for both NMI and Accuracy in the multi-case scenario as well. In fact, for the datasets we have experimented on, the plots suggest much less fluctuations and variations.

Note that our baseline MHC uses only cosine distance. Compared to this baseline, MVHC-OT with cosine distance almost always performs better or equally well for all values of t.

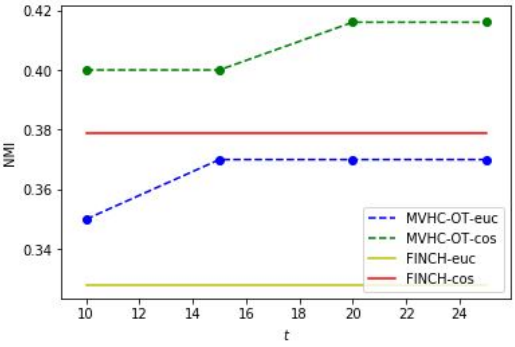
Similar to the single-view case, we can again use a different smaller set of values $\{10, 20, 25\}$ for t which would still enable us to get similar best performance. So a possible modification for the parameter selection step may be to use these two different sets for single and multi-view datasets separately to reduce the number of parameters to select from. However, our experiments are not conclusive that for other datasets the omitted values of t will not give optimal performance. Hence, we suggest to use the entire set of values during grid search for both the single-view and multi-view cases.



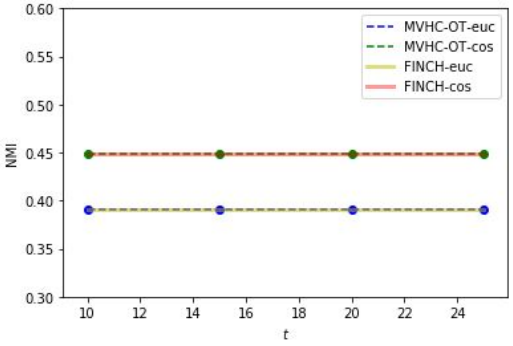
(a) Leukemia



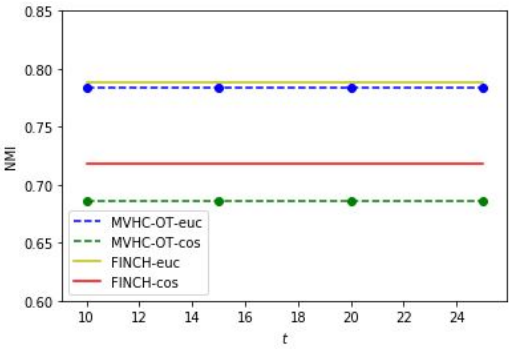
(b) Iris



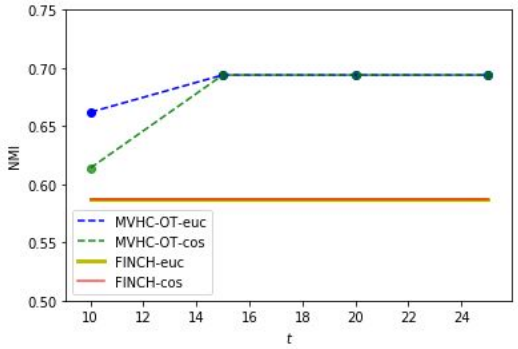
(c) Glass



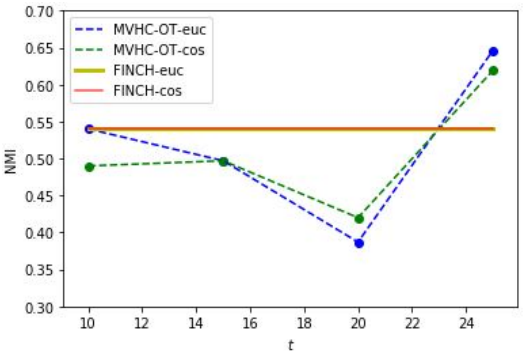
(d) Wine



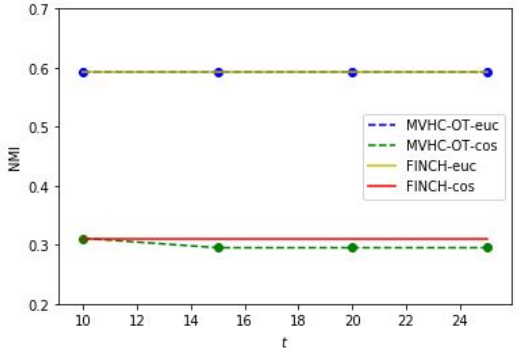
(e) Zoo



(f) Brain

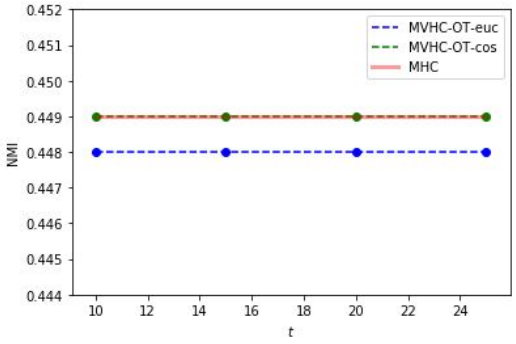


(g) SRBCT

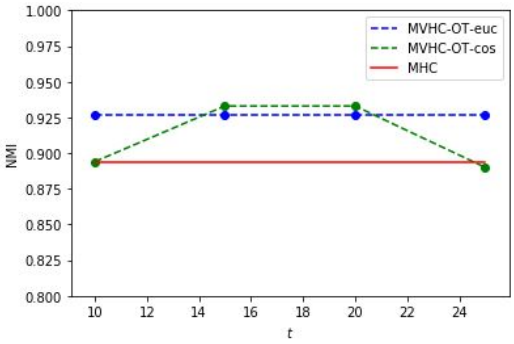


(h) Seeds

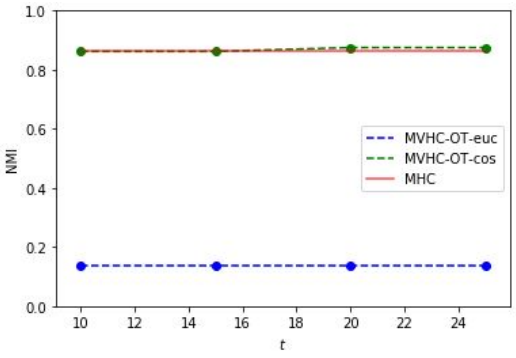
Figure 5.1: Plots illustrating sensitivity analysis for change in NMI with different values of threshold t and distance measure (euclidean/cosine) for single-view datasets when K is given



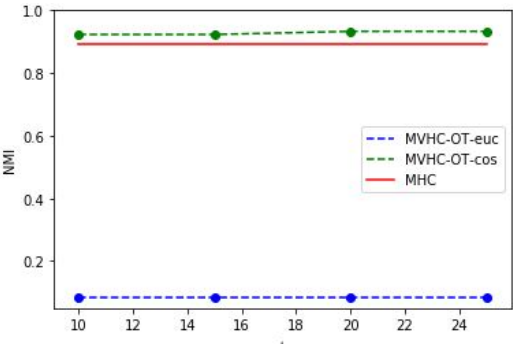
(a) Caltech101



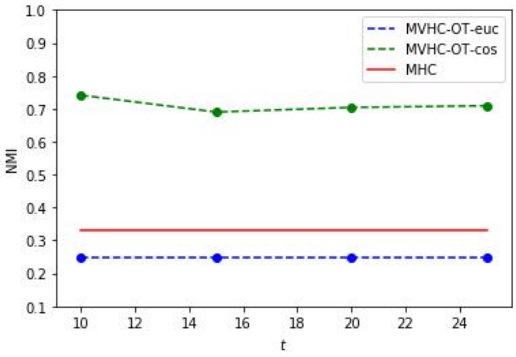
(b) UCI



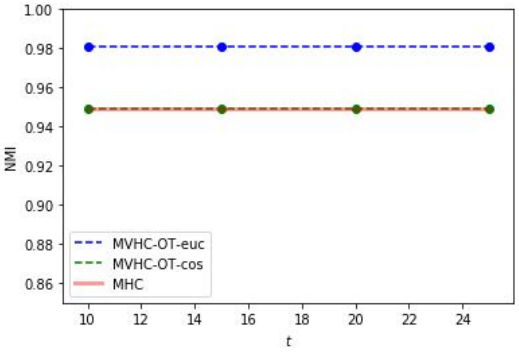
(c) Football



(d) Olympics

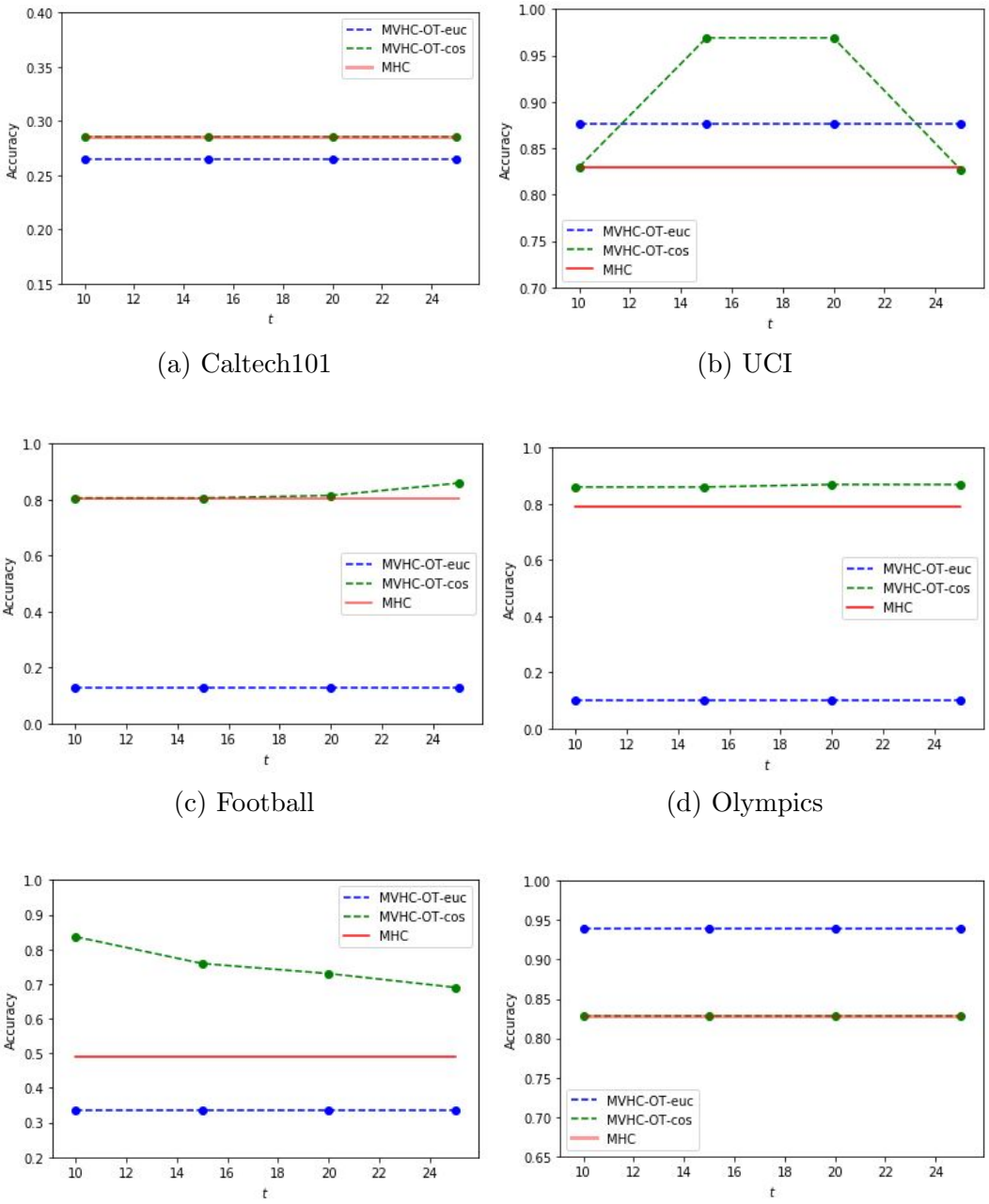


(e) Politicsie



(f) 100Leaves

Figure 5.2: Plots illustrating sensitivity analysis for change in NMI with different values of threshold t and distance measure (euclidean/cosine) for multi-view datasets when K is given



(a) Caltech101

(b) UCI

(c) Football

(d) Olympics

(e) Politicsie

(f) 100Leaves

Figure 5.3: Plots illustrating sensitivity analysis for change in Accuracy with different values of threshold t and distance measure (euclidean/cosine) for multi-view datasets when K is given

Chapter 6

Conclusion and Future Work

6.1 Conclusion

In this work we have developed a Multi-View Agglomerative Hierarchical Clustering technique called MVHC-OT using the concept of Optimal Transport (OT) which has been very recently used for the first time in the realm of hierarchical clustering by [23] for taking into consideration the entire distribution of the data, but in single-view domain. Also, incorporation of OT into naive agglomerative hierarchical clustering leads to very high time complexity.

First, we have incorporated OT into the single-view FINCH algorithm [20] called FINCH-OT, which merges multiple clusters in each iteration and as a result leads to very few iterations even for large datasets. This leads to a comparatively low time complexity even with the incorporation of OT. Then as an extension to FINCH-OT, we have leveraged OT in the multi-view hierarchical clustering framework proposed by [4]. Subsequently we have shown that our proposed multi-view clustering technique, MVHC-OT is a generalized version of the single-view FINCH-OT algorithm derived by us for single-view domain.

Elaborate experiments have also been performed on both real-life single-view and multi-view datasets which show that our proposed MVHC-OT algorithm outperforms or performs almost equally well when compared against other state-of-the-art algorithms on most of the datasets. In addition, unlike many existing multi-view clustering algorithms, our method is able to output a partitioning even when the required number of clusters for a dataset is not pre-specified. Experiments have shown that the number of clusters predicted by our algorithm is not too far off from the actual number of clusters in the provided dataset.

6.2 Scope for Future Work

In spite of showing promising performance on the real-life datasets we have experimented on, our proposed MVHC-OT Algorithm does suffer from shortcomings with respect to hyperparameter selection. MVHC-OT has three hyperparameters, the regularization constant (λ), the fraction of data samples to be considered as a threshold for deciding whether to use OT or traditional means for deciding inter-cluster distances (given by t, such that the fraction of samples is equal to n/t, where n = number of data samples) and the distance measure (cosine or euclidean) to be used in the cost matrix for OT. Among these three hyperparameters, we have

already mentioned in Section 5.2.4 that λ can be fixed to a particular value without change in performance, and hence may be ignored as a potential hyperparameter which needs to be tuned. But as far as the other two hyperparameters are concerned, future works devoted to choosing particular parameter values depending on some properties of the given dataset without actually going through the process of performing the algorithm on all the possible parameter values and then choosing the ones giving best performance can be interesting.

Also note that our proposed algorithm is an agglomerative hierarchical clustering approach. Other hierarchical clustering techniques for multi-view domain present in literature are also agglomerative in nature. To the best of our knowledge, Divisive/Top-down Hierarchical Clustering techniques have not been explored at all in the multi-view realm. This leads us to another possible direction of work in multi-view hierarchical clustering. Also, can OT be used as a metric to define which clusters to divide in case of divisive hierarchical clustering like we have previously seen for the agglomerative case for merging two clusters? Such questions are worth answering and may well be investigated in future works.

Furthermore, as mentioned in Section 5.2.6, OT seems to be resistant to the curse of dimensionality problem to some extent. Further studies may be performed to investigate this observation.

Bibliography

[1] C. Xu, D. Tao, and C. Xu, "A survey on multi-view learning," *arXiv preprint arXiv:1304.5634*, 2013.

[2] Y. Yang and H. Wang, "Multi-view clustering: A survey," *Big Data Mining and Analytics*, vol. 1, no. 2, pp. 83–107, 2018.

[3] C. Zhang, Q. Hu, H. Fu, P. Zhu, and X. Cao, "Latent multi-view subspace clustering," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4279–4287, 2017.

[4] Q. Zheng, J. Zhu, and S. Ma, "Multi-view hierarchical clustering," *arXiv preprint arXiv:2010.07573*, 2020.

[5] U. Von Luxburg, "A tutorial on spectral clustering," *Statistics and computing*, vol. 17, no. 4, pp. 395–416, 2007.

[6] H. Gao, F. Nie, X. Li, and H. Huang, "Multi-view subspace clustering," in *Proceedings of the IEEE international conference on computer vision*, pp. 4238–4246, 2015.

[7] Z. Kang, W. Zhou, Z. Zhao, J. Shao, M. Han, and Z. Xu, "Large-scale multi-view subspace clustering in linear time," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, pp. 4412–4419, 2020.

[8] R. Xia, Y. Pan, L. Du, and J. Yin, "Robust multi-view spectral clustering via low-rank and sparse decomposition," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 28, 2014.

[9] H. Wang, Y. Yang, and B. Liu, "Gmc: Graph-based multi-view clustering," *IEEE Transactions on Knowledge and Data Engineering*, vol. 32, no. 6, pp. 1116–1129, 2019.

[10] F. Nie, J. Li, X. Li, *et al.*, "Parameter-free auto-weighted multiple graph learning: a framework for multiview clustering and semi-supervised classification.," in *IJCAI*, pp. 1881–1887, 2016.

[11] Z. Zhang, L. Liu, F. Shen, H. T. Shen, and L. Shao, "Binary multi-view clustering," *IEEE transactions on pattern analysis and machine intelligence*, vol. 41, no. 7, pp. 1774–1782, 2018.

[12] X. Peng, Z. Huang, J. Lv, H. Zhu, and J. T. Zhou, "Comic: Multi-view clustering without parameter selection," in *International Conference on Machine Learning*, pp. 5092–5101, PMLR, 2019.

[13] W. Zhang, X. Wang, D. Zhao, and X. Tang, "Graph degree linkage: Agglomerative clustering on a directed graph," in *European Conference on Computer Vision*, pp. 428–441, Springer, 2012.

[14] W. Zhang, D. Zhao, and X. Wang, "Agglomerative clustering via maximum incremental path integral," *Pattern Recognition*, vol. 46, no. 11, pp. 3056–3065, 2013.

[15] D. Müllner, "Modern hierarchical, agglomerative clustering algorithms," *arXiv preprint arXiv:1109.2378*, 2011.

[16] D. Zhao and X. Tang, "Cyclizing clusters via zeta function of a graph," *Advances in Neural Information Processing Systems*, vol. 21, pp. 1953–1960, 2008.

[17] A. Fernández and S. Gómez, "Solving non-uniqueness in agglomerative hierarchical clustering using multidendrograms," *Journal of Classification*, vol. 25, no. 1, pp. 43–65, 2008.

[18] J. Yang, E. Grunsky, and Q. Cheng, "A novel hierarchical clustering analysis method based on kullback–leibler divergence and application on dalaimiao geochemical exploration data," *Computers & Geosciences*, vol. 123, pp. 10–19, 2019.

[19] D. M. Witten and R. Tibshirani, "A framework for feature selection in clustering," *Journal of the American Statistical Association*, vol. 105, no. 490, pp. 713–726, 2010.

[20] S. Sarfraz, V. Sharma, and R. Stiefelhagen, "Efficient parameter-free clustering using first neighbor relations," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8934–8943, 2019.

[21] H. Mirzaei, "A novel multi-view agglomerative clustering algorithm based on ensemble of partitions on different views," in *2010 20th International Conference on Pattern Recognition*, pp. 1007–1010, IEEE, 2010.

[22] C. Villani, *Optimal transport: old and new*, vol. 338. Springer, 2009.

[23] S. Chakraborty, D. Paul, and S. Das, "Hierarchical clustering with optimal transport," *Statistics & Probability Letters*, vol. 163, p. 108781, 2020.

[24] S. Bickel and T. Scheffer, "Multi-view clustering.," in *ICDM*, vol. 4, pp. 19–26, Citeseer, 2004.

[25] S. Sun, "A survey of multi-view machine learning," *Neural computing and applications*, vol. 23, no. 7, pp. 2031–2038, 2013.

[26] S. Dasgupta, M. L. Littman, and D. McAllester, "Pac generalization bounds for co-training," *Advances in neural information processing systems*, vol. 1, pp. 375–382, 2002.

[27] C. K. Reddy and B. Vinzamuri, "A survey of partitional and hierarchical clustering algorithms," in *Data clustering*, pp. 87–110, Chapman and Hall/CRC, 2018.

[28] S. A. Elavarasi, J. Akilandeswari, and B. Sathiyabhama, "A survey on partition clustering algorithms," *International Journal of Enterprise Computing and Business Systems*, vol. 1, no. 1, 2011.

[29] J. H. Ward Jr, "Hierarchical grouping to optimize an objective function," *Journal of the American statistical association*, vol. 58, no. 301, pp. 236–244, 1963.

[30] Y. Rani[1] and H. Rohil, "A study of hierarchical clustering algorithm," *ter S & on Te SIT*, vol. 2, p. 113, 2013.

[31] G. Monge, "Mémoire sur la théorie des déblais et des remblais," *Histoire de l'Académie Royale des Sciences de Paris*, 1781.

[32] C. Villani, *Topics in optimal transportation*. No. 58, American Mathematical Soc., 2003.

[33] L. V. Kantorovich, "On the translocation of masses," in *Dokl. Akad. Nauk. USSR (NS)*, vol. 37, pp. 199–201, 1942.

[34] M. Cuturi, "Sinkhorn distances: Lightspeed computation of optimal transport," *Advances in neural information processing systems*, vol. 26, pp. 2292–2300, 2013.

[35] P. A. Knight, "The sinkhorn–knopp algorithm: convergence and applications," *SIAM Journal on Matrix Analysis and Applications*, vol. 30, no. 1, pp. 261–275, 2008.

[36] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Communications of the ACM*, vol. 18, no. 9, pp. 509–517, 1975.

[37] D. B. Carr, C. J. Young, R. C. Aster, and X. Zhang, "Cluster analysis for ctbt seismic event monitoring," tech. rep., Sandia National Labs., Albuquerque, NM (US); Sandia National Labs ..., 1999.

[38] C. Romesburg, *Cluster analysis for researchers*. Lulu. com, 2004.

[39] F. J. Rohlf and D. R. Fisher, "Tests for hierarchical structure in random data sets," *Systematic Biology*, vol. 17, no. 4, pp. 407–412, 1968.

[40] W.-B. Xie, Y.-L. Lee, C. Wang, D.-B. Chen, and T. Zhou, "Hierarchical clustering supported by reciprocal nearest neighbors," *Information Sciences*, vol. 527, pp. 279–292, 2020.

[41] T. Zhang, P. Ji, M. Harandi, W. Huang, and H. Li, "Neural collaborative subspace clustering," in *International Conference on Machine Learning*, pp. 7384–7393, PMLR, 2019.

[42] L. Kaufman and P. J. Rousseeuw, *Finding groups in data: an introduction to cluster analysis*, vol. 344. John Wiley & Sons, 2009.

[43] T. R. Golub, D. K. Slonim, P. Tamayo, C. Huard, M. Gaasenbeek, J. P. Mesirov, H. Coller, M. L. Loh, J. R. Downing, M. A. Caligiuri, *et al.*, "Molecular classification of cancer: class discovery and class prediction by gene expression monitoring," *science*, vol. 286, no. 5439, pp. 531–537, 1999.

[44] S. L. Pomeroy, P. Tamayo, M. Gaasenbeek, L. M. Sturla, M. Angelo, M. E. McLaughlin, J. Y. Kim, L. C. Goumnerova, P. M. Black, C. Lau, *et al.*, "Prediction of central nervous system embryonal tumour outcome based on gene expression," *Nature*, vol. 415, no. 6870, pp. 436–442, 2002.

[45] A. I. Su, J. B. Welsh, L. M. Sapinoso, S. G. Kern, P. Dimitrov, H. Lapp, P. G. Schultz, S. M. Powell, C. A. Moskaluk, H. F. Frierson, *et al.*, "Molecular classification of human carcinomas by use of gene expression signatures," *Cancer research*, vol. 61, no. 20, pp. 7388–7393, 2001.

[46] J. Alcalá-Fdez, A. Fernández, J. Luengo, J. Derrac, S. García, L. Sánchez, and F. Herrera, "Keel data-mining software tool: data set repository, integration of algorithms and experimental analysis framework.," *Journal of Multiple-Valued Logic & Soft Computing*, vol. 17, 2011.

[47] D. Dua, C. Graff, *et al.*, "Uci machine learning repository," 2017.

[48] J. Jin, W. Wang, *et al.*, "Influential features pca for high dimensional clustering," *Annals of Statistics*, vol. 44, no. 6, pp. 2323–2359, 2016.

[49] L. Fei-Fei, R. Fergus, and P. Perona, "Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories," in *2004 conference on computer vision and pattern recognition workshop*, pp. 178–178, IEEE, 2004.

[50] A. Asuncion and D. Newman, "Uci machine learning repository," 2007.

[51] D. Greene and P. Cunningham, "Producing a unified graph representation from multiple social network views," in *Proceedings of the 5th annual ACM web science conference*, pp. 118–121, 2013.

[52] C. Mallah, J. Cope, and J. Orwell, "Plant leaf classification using probabilistic integration of shape, texture and margin features," *Signal Processing, Pattern Recognition and Applications*, vol. 5, no. 1, pp. 45–54, 2013.

[53] K. Florek, J. Łukaszewicz, J. Perkal, H. Steinhaus, and S. Zubrzycki, "Sur la liaison et la division des points d'un ensemble fini," in *Colloquium mathematicum*, vol. 2, pp. 282–285, 1951.

[54] R. Sibson, "Slink: an optimally efficient algorithm for the single-link cluster method," *The computer journal*, vol. 16, no. 1, pp. 30–34, 1973.

[55] D. Defays, "An efficient algorithm for a complete link method," *The Computer Journal*, vol. 20, no. 4, pp. 364–366, 1977.

[56] R. R. Sokal, "A statistical method for evaluating systematic relationships.," *Univ. Kansas, Sci. Bull.*, vol. 38, pp. 1409–1438, 1958.

[57] L. L. McQuitty, "Similarity analysis by reciprocal pairs for discrete and continuous data," *Educational and Psychological measurement*, vol. 26, no. 4, pp. 825–831, 1966.

[58] P. Legendre and L. Legendre, *Numerical ecology*. Elsevier, 2012.

[59] G. Liu, Z. Lin, S. Yan, J. Sun, Y. Yu, and Y. Ma, "Robust recovery of subspace structures by low-rank representation," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 1, pp. 171–184, 2012.

[60] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, *et al.*, "Scikit-learn: Machine learning in python," *the Journal of machine Learning research*, vol. 12, pp. 2825–2830, 2011.

[61] R. Flamary, N. Courty, A. Gramfort, M. Z. Alaya, A. Boisbunon, S. Chambon, L. Chapel, A. Corenflos, K. Fatras, N. Fournier, L. Gautheron, N. T. Gayraud, H. Janati, A. Rakotomamonjy, I. Redko, A. Rolet, A. Schutz, V. Seguy, D. J. Sutherland, R. Tavenard, A. Tong, and T. Vayer, "Pot: Python optimal transport," *Journal of Machine Learning Research*, vol. 22, no. 78, pp. 1–8, 2021.

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

[63] R. Sinkhorn, "Diagonal equivalence to matrices with prescribed row and column sums," *The American Mathematical Monthly*, vol. 74, no. 4, pp. 402–405, 1967.

Appendix A

Sinkhorn's Algorithm

Here we show in detail how to solve Equation (2.4) using Sinkhorn's Algorithm. For clarity, Equation (2.4) is as follows:

$$\min_{\gamma}\{\sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}\gamma_{ij} + \lambda\sum_{i=1}^{m}\sum_{j=1}^{n}\gamma_{ij}log(\gamma_{ij})\} \text{ such that } \gamma 1_n = a, \gamma^T 1_m = b \tag{A.1}$$

For this section, we introduce the following notations:
$M_{XY} = [C(x_i, y_j)]_{ij}$,
$U(a,b) = \{\gamma \in \mathbb{R}_+^{m\times n} | \gamma 1_n = a, \gamma^T 1_m = b\}$

In vector notation, Equation (A.1) can now be written as:

$$\min_{\gamma\in U(a,b)} < \gamma, M_{XY} > +\lambda \sum_{i,j=1}^{mn} \gamma_{ij} log\gamma_{ij} \tag{A.2}$$

where $< A, Z > = \sum_{ij}^{mn} A_{ij}Z_{ij}$ gives the Frobenius Inner Product of two matrices A and Z of the order $m \times n$.

Proposition 1. *If* $\gamma^\lambda \stackrel{def}{=} \underset{\gamma\in U(a,b)}{argmin} < \gamma, M_{XY} > +\lambda\sum_{i,j=1}^{mn}\gamma_{ij}log\gamma_{ij}$, *then* $\exists\ u \in \mathbb{R}_+^m$, $v \in \mathbb{R}_+^n$, *such that* $\gamma^\lambda = diag(u)Bdiag(v)$, $B \stackrel{def}{=} e^{-M_{XY}/\lambda}$.
Then $d_M^\lambda(\mu,\nu) =< \gamma^\lambda, M_{XY} >$ *gives the solution to Equation (A.2)*

Using Lagrange Multipliers we can get the Lagrangian from Equation (A.2) as follows:

$$L(\gamma, \alpha, \beta) = \sum_{ij}^{mn} \gamma_{ij}M_{ij} + \lambda\gamma_{ij}log\gamma_{ij} + \alpha^T(\gamma 1 - a) + \beta^T(\gamma^T 1 - b) \tag{A.3}$$

Taking derivative of L by γ_{ij} and setting it to 0, we get

$$\begin{aligned}
&\frac{\partial L}{\partial \gamma_{ij}} = M_{ij} + \lambda(\gamma_{ij}\frac{1}{\gamma_{ij}} + log\gamma_{ij}) + \alpha_i + \beta_j \\
&\implies 0 = M_{ij} + \lambda(1 + log\gamma_{ij}) + \alpha_i + \beta_j \\
&\implies \gamma_{ij} = e^{-1/2-\alpha_i/\lambda} e^{-M_{XY}/\lambda} e^{-1/2-\beta_j/\lambda} \\
&\implies \gamma_{ij} = u_i B_{ij} v_j
\end{aligned}$$

Now B is strictly positive. Given this, Sinkhorn's Theorem [63] states that there must exist a matrix γ^λ of the form $diag(u)Bdiag(v)$ which is unique in nature and belongs to $U(a, b)$ where u, $v \geq 0$.
Hence, we have the following:

$$\gamma^\lambda \in U(a,b) \iff \begin{cases} diag(u)Bdiag(v)1_n = a \\ diag(v)B^T diag(u)1_m = b \end{cases} \tag{A.4}$$

Here, $diag(u)B = u \odot B$, $diag(v)1_n = v$, $diag(v)B^T = v \odot B^T$ and $diag(u)1_m = u$, where $\odot$ denotes element-wise product.

Then, we have

$$\gamma^\lambda \in U(a,b) \iff \begin{cases} u \odot Bv = a \\ v \odot B^T u = b \end{cases} \tag{A.5}$$

$$\gamma^\lambda \in U(a,b) \iff \begin{cases} u = a/Bv \\ v = b/B^T u \end{cases} \tag{A.6}$$

Sinkhorn's Algorithm simply asks to iteratively update u and v using Equation (A.6) in an alternate manner. In very simple terms, Sinkhorn's Algorithm is depicted in Algorithm 11.

Algorithm 11: Sinkhorn's Algorithm

Data: B, a, b
Result: u, v
while *not converged* **do**
  $u = a/Bv$
  $v = b/B^T u$
end

Then let us assume that after L iterations Algorithm 11 returns u_L and v_L. Then we can get,

$$\gamma_L = diag(u_L)Bdiag(v_L)$$

And, the solution is given by

$$< \gamma_L, M_{XY} >= u_L^T (B \odot M_{XY}) v_L$$