Hierarchical Shape Retrieval on the Unit Hypersphere

Glizela Taino[1], Yixin Lin[2], Mark Moyou[3], and Adrian M. Peter[4]

[1] *Hawaii Pacific University, Honolulu, Hawaii*
[2] *Duke University, Durham, North Carolina*
[3] *Florida Institute of Technology, Melborne, Florida*
[4] *Florida Institute of Technology, Melborne, Florida*

June 3, 2016

Abstract

This paper outlines our experiemental implementation in approaching shape retieval by exploring different ways to cluster on a unit hypersphere, testing algoritm complexity between divisive and agglomerative hierarchical clustering, and how it is used to execute a model to model retrieval. We create the feature representation of each shape in the data set with Laplace-Beltrami Operator signatures and their probability density coefficients which map onto the surface of a unit hypersphere. We take advantage of this geometry to recursively cluster the observations under various prototypes and create a hierarchy based on centroid linkage. This essentially builds a decision tree, so that, given a query, we preprocess it in the same way, map it onto the manifold, and perform a model to model comparison to decide which branches to traverse in order to get back the most relevant results from the data base. We illustrate the effectiveness of our approach by testing our retrieval results under metrics from the Shape Retrieval Contest (SHREC). Our results show _____. In conclusion _____.

Introduction

In visually understanding the world around us, we have attributes such as color, texture, scale, ect. But to truly understand objects, all we fundamentally need is shape. Understanding shape is essential in understanding the world and it's meaning. Our goal is to allow computers to understand shape like humans naturally do. We want them to see through all the distortions and capture the essence of an object as an important piece of the quest to make machines more intelligent. The question of shape retrieval is: given a shape, how can we find the most similar shapes from our database? How can we understand these shapes using geometry alone, without any textual metadata or context?

In the big picture, shape retrieval can be broken up into three stages: shape representation, feature representation, and retrieval mechanics. The shape is given to us as 2D points, 3D points, or 3D mesh. These shapes undergo preprocessing in order to transform into a feature representation that can be used as a measureable property for clustering similar objects. For our particular implementation we used Laplace-Beltrami Operator signatures and wavelet density coefficients to map that onto the surface of a unit hypersphere. Our main work for this paper so far is retrieval mechanics.

To understand how to best cluster on the multidimensional unit hypersphere's curved manifold, we exectued an experiement with sample data plotted by the Von-Mises distribution on a 2-sphere. Then we ran a MATLAB implementation Sphereical K-means to test its accuracy on our sample data. We ran this clustering method against a method from differential geometry called the Karcher mean. Once we decided on a clustering method, we used this in hierarchical clustering to form a decision tree. We performed another experiment to decided between agglomerative and divisive hierarchical clustering. We tested its runtime, storage, and overall complexity of both approaches. We also analyzed both algorithms and found interesting results that we will expand on later in the paper.

Once we had working code to cluster on the unit hypersphere and created a decision tree, we researched how our algorithm and retrieval list will be evaluated by the Shape Retrieval Contest (SHREC). Unfortunately, our evaluation scores for all the metrics weren't ideal. We feel this may be due to the highly condensed area of data points which the clustering fails to appropriately group.

Clustering on a Sphere

In this investigation, we set out to test the Spherical K-means algorithm MATLAB implementation. To do this we had to understand the Von-Mises Fisher Distribution in order to create sample data on a sphere, as well as the K-means algorithm to further expand and visualize our understanding onto a higher, multidimensional space.

Von Mises distribution

Intuitively, the Von Mises distribution [9] is a simple approximation for the normal distribution on a circle (known as the *wrapped normal distribution*). The Von Mises probability density function is defined by the following equation:

$$P(x) = \frac{e^{b\cos(x-a)}}{2\pi I_0(b)}$$

where $I_0(x)$ is the modified Bessel function of the first kind; the Von Mises cumulative density function has no closed form.

The mean $\mu = a$ (intuitively, the angle that the distribution clustered around), and the circular variance $\sigma^2 = 1 - \frac{I_1(b)}{I_0(b)}$ (intuitively, b is the "concentration" parameter). Therefore, as $b \to 0$, the distribution becomes uniform; as $b \to \infty$, the distribution becomes normal with $\sigma^2 = 1/b$.

Von Mises-Fisher distribution

The Von-Mises Fisher distribution is the generalization of the Von Mises distribution to n-dimensional hyperspheres. It reduces to the Von-Mises distribution with $n = 2$. The probability density function is defined by the following equation:

$$f_p(\boldsymbol{x}; \boldsymbol{\mu}, \kappa) = C_p(\kappa) \exp(\kappa \boldsymbol{\mu}^T \boldsymbol{x})$$

where

$$C_p(\kappa) = \frac{\kappa^{p/2-1}}{(2\pi)^p I_{p/2-1}(\kappa)}$$

and intuitively is an approximation for the normal distribution on the hypersphere.

K-means clustering

k-means clustering is a simple and popular algorithm for the *unsupervised clustering problem*, the task of grouping a set of observations so that a group is "similar" within itself and "dissimilar" to other groups. k-means partitions n observations into k clusters, with each observation belonging to the nearest mean of the cluster. This problem is NP-hard in general, but there are heuristics which guarantee convergence to a local optimum.

The standard heuristic (known as *Lloyd's algorithm* [6]) is the following:

Algorithm 1 Lloyd's algorithm for k-means clustering

1: generate an initial set of k means
2: **while** not converged **do**
3: assign all data points to nearest Euclidean-distance mean
4: calculate new means to as the centroids of the observations in the cluster
5: **end while**

There is a choice of initialization method. The *Forgy method* randomly picks k observations as initial means, while the *Random Partition method* randomly picks a cluster for each observation.

The Lloyd's algorithm is a heuristic, so it does not guarantee a global optimum. Furthermore, there exists sets of points in which it converges in exponential time. However, it has been shown to have a smoothed polynomial running time, and in practice converges quickly.

Spherical k-means clustering

Spherical k-means clustering is the same idea, however, the points are not on a flat euclidean space but rather on the hypersphere. We investigated a MATLAB implementation by Nguyen [8, 7], which required a mean-and-norm-normalized dataset located on a hypersphere. Important aspects of this implementation include:

- When there exists an empty cluster, the largest cluster is split
- Use the dot product as "negative distance", which leverages the fact that observations are unit vectors on the hypersphere
- Use the normalized sum of observations as a centroid/mean, which leverages the fact that observations are unit vectors on the hypersphere. Note that this fails on pathological cases where the sum of observations is zero.

Our investigation

In order to test how well the spherical k-means clustering algorithm worked, we constructed a random data set using the Von Mises distribution random sampling function from the MATLAB Circular Statistics Toolbox [1]. We constructed 70 clusters of 10 points each, using random points on the sphere as means and with step size $\kappa = 100$, and then applied the spherical k-means clustering and visualized the results on the unit three dimensional sphere, color coded for each cluster and with estimated means labeled as red X's and true means labeled as black X's.

Karcher Mean

The Karcher mean is a geometric mean of various matrices and an extension of the well known geometric mean of two matrices. It is also referred to as the Riemannian geometric mean. In our investigation, we used it as a way to compute the average distance between two points on a hypersphere while obeying the curvature of the manifold. The process is stated in Algorithm 1. In order to do this we must incorporate a Logarithm map and Exponential map on the manifold.

The Logarithm map is defined as a function that takes a point, ρ_2, on the manifold and maps its projection vector, γ, on the tangent plane at an origin point, T_{p_1}.

$$\rho_2 = Exp_{\rho_1}(\gamma) = cos(|\gamma|)\rho_1 + sin(|\gamma|)\frac{\gamma}{|\gamma|} \tag{1}$$

On the first iteration, ρ_1 is the origin point of the tangent space.

The Exponential map can be thought of as the inverse function of the Logarithm map. It is a function that takes in a vector on the tangent plane at origin

Figure 1: Plot of 70 clusters and means on a 3-D sphere

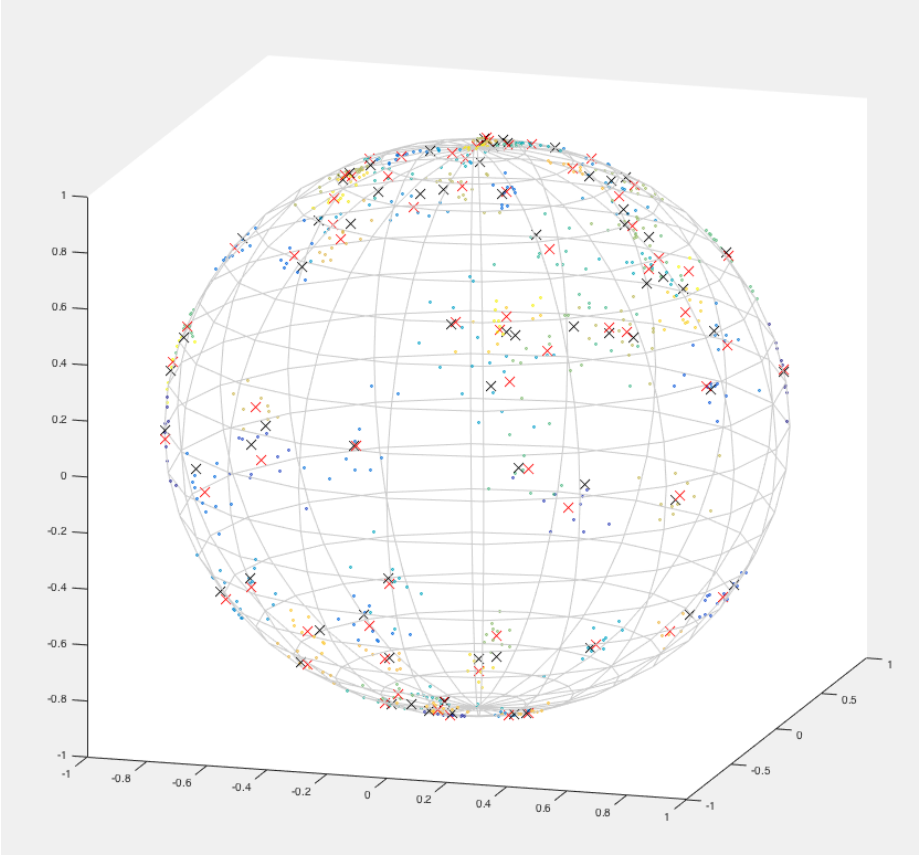
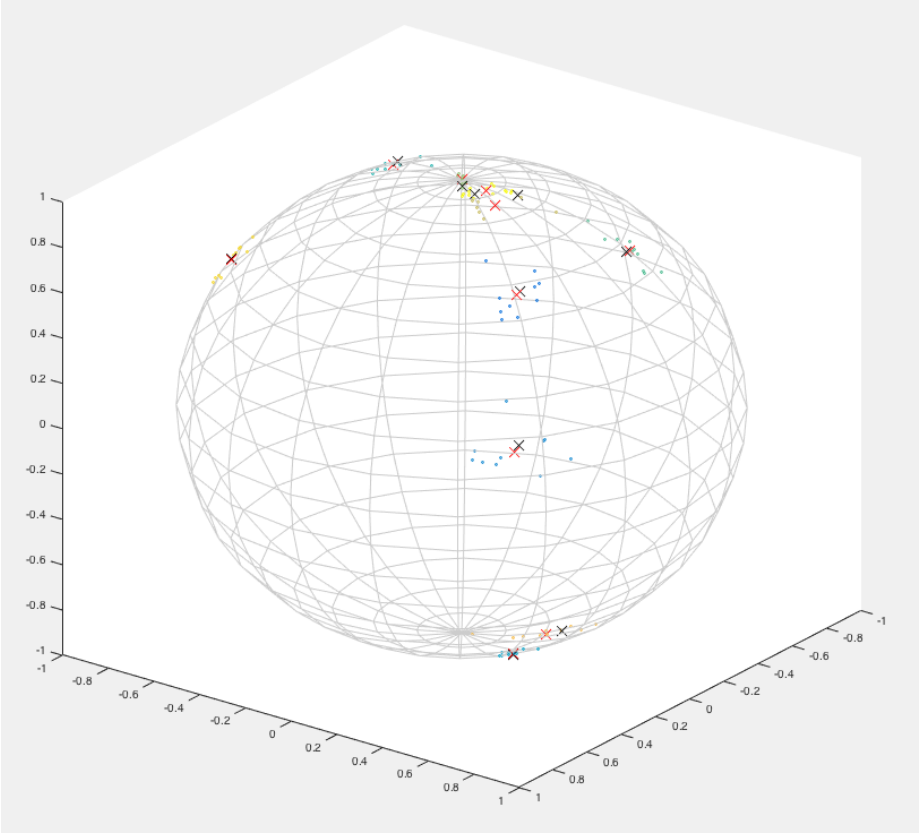


Figure 2: Plot of 10 clusters and means on a 3-D sphere



point and maps it on the hypersphere. As the difference between the updating vectors on the tangent space begin to show little change, it is thought as the optimal mean between the two points on the manifold.

$$\gamma = Log_{\rho_1}(\rho_2) = \tilde{\rho}\frac{cos^{-1}(\langle \rho_1, \rho_2 \rangle)}{\sqrt{\langle \tilde{\rho}, \tilde{\rho} \rangle}} \tag{2}$$

where $\tilde{\rho} = \rho_2 - \langle \rho_2, \rho_1 \rangle \rho_1$. Karcher mean, as we have mentioned before, is calculated in a way to obey the curvature of the hypershere.

Spherical mean

Unlike the Karcher mean, the spherical mean does not go along the curvature of the hypersphere. In the assignment step of this algorithm, where we adjust the cluster centers, we find the mean by summing up all the vectors in a cluster and normalizing it back onto the manifold.

$$x_l = \frac{\sum_{i \in X_l} \rho_i}{||\sum_{i \in X_l} \rho_i||}, l = 1 \dots K$$

where X_h are the center of each K number of clusters. This is the normalized gravity center of the new cluster.

Our Implementation

To find the best approach in calculating a cluster center, we put Karcher mean and Spherical K-mean to the test. We tested them in five types of situations in where the angles are either identical, orthogonal, opposite, two random angles, or multiple random angles as shown in Figure 3.

We found that Karcher mean and Spherical K-means produced the same results as long as the step size for Karcher mean was set to one. But both approaches failed in the case where the angles were opposite.

Hierarchical clustering

For our lab, we want to develop the most efficient way to group data based based on a measure of dissimilarity. We explored the topic of hierarchical clustering which is used in data mining and statistics. It is visualized useing a dendogram to show the organization and relationship among clusters. Hierarchical clustering can be implemented in two ways: divisive and agglomerative. We will elaborate on these two variants in this paper to compare the its algorithmic complexity, or how much resources are needed when computed, and the different types of cluster linkage criterion. Further information on cluster analysis can be found in [5].

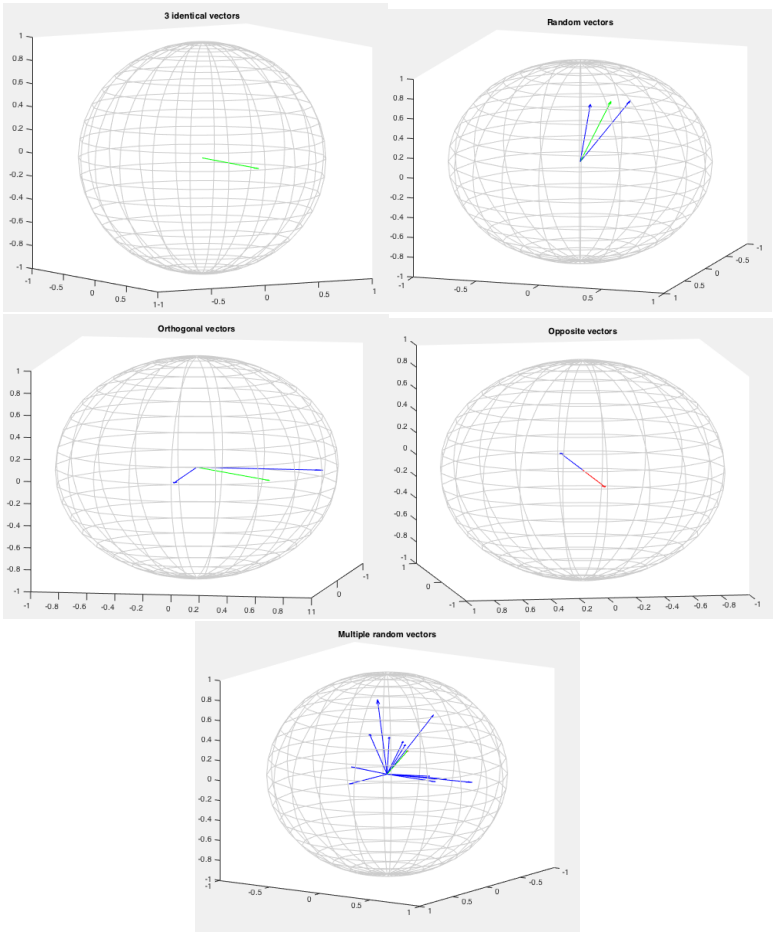


Figure 3: Different test cases comparing accuracy of Karcher mean and Spherical K-means.

Divisive and Agglomerative

Divisive clustering is also known as the "top down" method. This takes in a data set as one large cluster. Then as it moves down the hierarchy, it recusively splits until it reaches the leaves of the tree, i.e. each observation is its own singleton cluster.

Agglomerative clustering can be thought to do the opposite of divisive clustering and is known as a "bottom up" strategy. It takes in a data set and it initially looks at each observation as its own singleton cluster. Then, based on its linkage criterion, it will merge recursively to form new clusters until all the data is in one large cluster.

A dendrogram can be used in both cases to record and visualize the hierar-

chical nature of the clusters and their distance to each other.

Cluster Dissimilarity Measurements

In forming clusters, a measurement must be made to determine which clusters split or merge in regards to divisive or agglomerative clustering, respectively. There are many different ways to measure dissimilarity but this paper will explore single linkage, complete linkage, and centroid linkage. Each of the measurements below refer to ways in measuring distances between k number of clusters $c_{i=1...k}$ in relationship with specific points $x_{i=1...k}$ in those clusters.

Single Linkage

Single linkage takes the distance between the closest observations, or minimum distance, in each cluster and merges the those with the shortest distance. This works well for data sets that desire to cluster together long chained data points. Let $D(c_i, c_j)$ be the distance between two centroids for $i, j \in \{1, 2, ...k\}$, and $D(x_i, x_j)$ be the distance between two data points in $i, j \in \{1, 2, ...n\}$. Then single linkage defines the following relationship:

$$D(c_1, c_2) = \min_{x_1 \in c_1, x_2 \in c_2} D(x_1, x_2)$$

Complete Linkage

Complete linkage is similar to single linkage. It takes the distance between the farthest observations, or the maximum distance, in each cluster. The clusters that have the shortest measured distance are merged together. This measurement is ideal for data that is spherically organized relative to each other. The relationship is defined as

$$D(c_1, c_2) = \max_{x_1 \in c_1, x_2 \in c_2} D(x_1, x_2)$$

Centroid Linkage

Using centroid linkage involves taking the average of all points in a cluster and assigning that value as the mean cenroid. Then the centroids with the shortest distances are merged together. Centroid linkage is defined as

$$D(c_1, c_2) = D((\frac{1}{|c_1|} \sum_{x \in c_1} \vec{x}), (\frac{1}{|c_2|} \sum_{x \in c_2} \vec{x}))$$

Our Implementation

We developed both hierarchical divisive and agglomerative clustering functions to test which would organize data the fastest. In our particular application,

instead of flat clustering, we had to cluster over the curved surface of a sphere. In order to do this, we utilized the SPKmeans function [7]. We also chose our linkage criterion to be based on centroid linkage.

Essentially, we developed the divisive clustering by following the intuition mentioned above. For our agglomerative clustering implementation, we did a slightly different variation hoping its run time would be slightly better than the traditional method. In our code, intuitively, we initiated each point as its own cluster with its own centroid. Then we recursively merged two clusters together based on centroid linkage to form a new cluster with its own cluster centroid everytime. This process stops when there is a single point to represent the entire data's average centroid.

We tested with a relatively small, three dimensional data set of 150 points. In both divisive and agglomerative, we input different cluster values into SPKmeans which determines how often the function gets called. In the divisive method, we asked SPKmeans to return two clusters. This means the number of times we call SPKmeans is based on how many nodes we have which is $m = 2n - 1$ where n is the number of data points. For the agglomerative case, we asked SPKmeans to return the $n/2$ clusters. This means means we had to call SPKmeans at every level which is $l = \lceil \log_2 n \rceil$.

As a benchmark, we also used the MATLAB `linkage` function to construct a pairwise-distance based tree (merging nodes together by single-linkage) and compared the results.

Implementation details

In k-means, there is a choice of initial mean initialization. This affects the initial distribution of means and therefore potentially the number of iterations until convergence, but also may be expensive computationally. For instance, a good setting of the initial means will reduce the likelihood of empty clusters found, which is dealt with by splitting the largest cluster and is therefore computationally expensive.

For the agglomerative algorithm, it makes sense to spend time initializing the means (we chose to maximize the distance between the initial means), since the number of calls to SPKMeans is small and the number of clusters for each call is high. For the divisive algorithm, it doesn't make sense to incur this expense when SPKMeans is called many times with only the branching-factor number of clusters (generally a small constant).

This optimization and lack thereof, respectively, reduces the running time for both algorithms substantially.

Algorithm complexity analysis

Let n be the number of data points, k be the number of clusters (number of means), d be the dimensionality of the data, and i be the number of iterations until convergence. The runtime of the k-means algorithm (including spherical k-means) is $O(nkdi)$.

Let us assume that the cost of initializing the means and splitting empty clusters is negligible, i.e. $O(1)$. In practice, the two are related: we find that explicitly picking good starting means for the agglomerative algorithm (e.g. randomly, or by maximizing distance between the means) makes the probability of empty clusters small, whereas it becomes excessive and unnecessary for the divisive algorithm. See the **Implementation details** section for more discussion.

We will also assume that d and i are constants. Though in degenerate data sets i can be an exponential function of n, in practice i is a small constant on datasets with clustering structuring.

Let us also assume that we choose a branching factor of 2, i.e. a binary hierarchical tree structure. The following analysis does not depend on the branching factor, assuming it is constant.

Theorem. ***Agglomerative complexity***

$$T_{aggl}(n) = \Theta(n^2 di)$$

Proof. Our agglomerative algorithm has a recurrence relation of the following form:

$$T_{aggl}(n) = T(\frac{n}{2}) + nkdi$$

for some constant c, since we divide the problem size into two for every recursive call. But we choose $k = \frac{n}{2}$ each time, since the number of means we choose to cluster with is also half of n.

We use the master theorem [3] to solve this recurrence relation:

Master theorem variable	**Value**
a	1
b	2
$f(n)$	$\frac{n^2 di}{2}$
c	2
$log_b(a)$	0

This satisfies case 3 of the master theorem:

$$f(n) \in \Omega(n^c) \text{ s.t. } c > log_b(a)$$

since $f(n) \in \Omega(n^2)$ s.t. $c = 2 > log_b(a) = 0$, and

$$af(\frac{n}{b}) \leq k_0 f(n) \text{ for some } k_0 < 1 \text{ and sufficiently large } n$$

since when $k_0 = \frac{1}{4}$, $\left\{ f(\frac{n}{2}) = (\frac{1}{4})\frac{n^2 di}{2} \right\} \leq \left\{ k_0 f(n) = (\frac{1}{4})\frac{n^2 di}{2} \right\}$

and so, by the master theorem: $T_{aggl}(n) = \Theta(n^2 di)$. $\square$

Theorem. ***Divisive complexity***

$$T_{div}(n) = \Theta(n \log(n) di)$$

Proof. Our divisive algorithm has a recurrence relation of the following form:

$$T_{div}(n) = 2T(\frac{n}{2}) + nkdi$$

for some constant c, since we divide the problem size into two for every recursive call and we have two subproblems each time. But we choose $k = 2$ each time, since the number of means we choose to cluster with is always the branching factor.

We use the master theorem [3] to solve this recurrence relation:

Master theorem variable	**Value**
a	2
b	2
$f(n)$	$n(2)di$
c	1
$log_b(a)$	1

This satisfies case 2 of the master theorem:

$$f(n) \in \Theta(n^c \log_{k_0} n) \text{ s.t. } c = log_b(a)$$

by letting $k_0 = 0$, since $f(n \log^0(n)) = f(n) \in \Theta(n)$ s.t. $c = 1 = log_b(a) = 1$. and so, by the master theorem: $T_{div}(n) = \Theta(n \log(n) di)$. □

Linear assignment

The assignment problem is a classical problem in graph theory and combinatorial optimization[2]. Informally, it deals with finding the most optimal (least-cost) way of assigning tasks to workers. It also has an equivalent graph theory formulation. We call such an assignment problem problem *linear* if there are equal numbers of tasks and workers and the total cost for all tasks is the sum of the individual costs incurred. This is a quite generalized problem that has many applications in resource allocation, including originally in transportation theory.

Formal specification

Let X and Y be two sets of equal size. Let C be a cost function

$$C : X \times Y \to \mathbb{R}$$

The problem is to find a bijection $f : A \to T$ such that the cost function is minimized:

$$\sum_{x \in X} C(x, f(x))$$

The linear assignment problem is often given as a cost matrix

$$C \mid c_{ij} = \text{cost of moving element } i \text{ to } j$$

Equivalently, in graph theoretic terms: given a weighted bipartite graph, find the minimum weight perfect matching.

Motivations

This problem first arose in transportation theory, where the goal is to assign n mines to n factories which consume the ore that the mines produce. However, the problem arises whenever there is a need to assign tasks to workers with potentially different costs for each potential task-worker pair.

We investigate this problem in order to align the discrete probability distributions of two shapes and then measure the distance between them. This is explained in the **sliding wavelets** section.

Algorithms and runtime

Because this problem can be formulated as a linear program, it can be solved with general linear program solving algorithms (e.g. simplex and variants). However, we can solve this more efficiently by exploiting the structure of the problem.

One famous algorithm is the famous Hungarian algorithm, which was proposed by Kuhn in the 1950s and originally had time complexity of $O(n^4)$, although this was improved by Lawler in 1976 to $O(n^3)$.

We use the Jonker-Volgenant algorithm[4], which is notably faster in a practical sense.

References

[1] Phillip Berens. Matlab central file exchange: Circular statistics toolbox. *http://www.mathworks.com/matlabcentral/fileexchange/10676-circular-statistics-toolbox–directional-statistics-*. Accessed: 2016-05-18.

[2] Rainer E Burkard and Eranda Cela. *Linear assignment problems and extensions*. Springer, 1999.

[3] Thomas H.. Cormen, Charles Eric Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*, volume 6. MIT press Cambridge, 2001.

[4] Roy Jonker and Anton Volgenant. A shortest augmenting path algorithm for dense and sparse linear assignment problems. *Computing*, 38(4):325–340, 1987.

[5] Leonard Kaufman and Peter J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. John Wiley & Sons, Inc., 2008.

[6] Stuart P. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2):129–137, March 1982.

[7] Vinh Nguyen. Matlab central file exchange: The spherical k-means algorithm. *http://www.mathworks.com/matlabcentral/fileexchange/32987-the-spherical-k-means-algorithm*. Accessed: 2016-05-18.

[8] Vinh Nguyen. Gene clustering on the unit hypersphere with the spherical k-means algorithm: coping with extremely large number of local optima. In *World Congress in Computer Science, Computer Engineering, and Applied Computing (Hamid R. Arabnia and Youngsong Mun 14 July 2008 to 17 July 2008)*, pages 226–233. CSREA Press, 2008.

[9] Eric W. Weisstein. von mises distribution. *http://mathworld.wolfram.com/vonMisesDistribution.html*. Accessed: 2016-05-18.