Background Information of Bregman Hard Clustering:

Bregman Divergence:

In simple terms, with convex function f: $R_n \rightarrow R$, which is typically derivable, the Bregman divergence $B_f(\cdot, y)$ is the gap that exists between f and its linearization at y.

Distortion Rate Function

This is the achieved distortion for a given rate.

Bregman Information:

Given a random variable X that takes place in a finite set $C = \{xi\}^{m}_{i=1}$ which follows the discrete probability measure v. As such, we will let the distortion be measured by the Bregman divergence represented as dφ. So with a simple encoding scheme S which represents the random variable with a constant vector, the solution to the distortion rate here is the trivial assignment, where the distortion rate function is represented as $E_v[d\varphi(X,S)]$. To optimize this, we would need to select the correct variables. This function represents the Bregman information of X and is depicted as $I\varphi(X)$

Bregman Representative:

The optimal vector S, which achieves the minimal distortion is the Bregman representative of X. $I\varphi(X) = \min_{s\in ri}(S)\ Ev[d\varphi(X,S)]$. Based on Bregman clustering theorems, this representative always exists, and the minimizer is the expectation of the random variable X.

Bregman Clustering Formula:

When X has a large Bregman information, it is best to encode X with multiple representatives as a lower quantization error is better. In this case, it is best to split the set X into k disjoint partitions, where each has its own Bregman representative. In this case a random value M over the partitioned representatives serves as an adequate quantization factor for X. $M = \{\mu h\}^{k}_{h=1}$ denote the set of partitioned representatives, as such we have $\pi = \{\pi_h\}\ k_h=1$ with $\pi_h = \sum_{xi\in Xh} v$ which depicts the induced probability measure on M. The expectation is calculated over X such that $E_X[d\varphi(X,M)] = {}^{k}\sum_{h=1} \sum_{xi\in Xh} v_i d\varphi(x_i, \mu_h)$.

Bregman Hard Clustering Algorithm:

Input: X, v, dφ, k (Number of clusters)

Output: M^t, local minimizer of $L\varphi(M) = {}^{k}\sum_{h=1} \sum_{xi\in Xh} v_i d\varphi(x_i, \mu_h)$ where $M = \{\mu_h\}^{k}_{h=1}$ and the hard partitioning of X.

Method:

```
First we must initialize $\{\mu_h\}^{k}_{h=1}$ with $\mu_h \in ri(S)$

//Assignment
Set $X_h \leftarrow 0,\ 1 \le h \le k$
For i = 1 to m do
$X_h \leftarrow X_h \cup \{x_i\}$
```

```
    where h = $h^t(x_i) = \text{argmin}^{h'} d\varphi(x_i, \mu_{h'})$
    End for loop

    //Reestimate
    For h = 1 to k do
    $\pi_h \leftarrow \sum_{x_i \in X_h} v_i$
    $\mu_h \leftarrow 1/\pi_h \sum_{x_i \in X_h} v_i x_i$
    End for

    Repeat until convergence is achieved
    return $M^t \leftarrow \{\mu_h\}^k_{h=1}$
```

In simplest terms Bregman's hard clustering undergoes 3 phases, initialization, assignment, and reestimation. In essence, the assignment phase determines the closest nodes relative to the centroid, and compiles them as such, from nearest to furthest from the given centroid. This then means we are required to put a limit on how many nodes can fit into a cluster.

Idea Behind Fuzzy Bregman's Clustering

In core functionality the Bregman's Clustering algorithm continually takes a centroid, creates assignments, and re estimates the centroid until convergence has been achieved. The benefits to Bregman's Clustering are its scalability being similar to KMean's, and its simplicity in pair with its effectiveness in minimizing loss. One downside to this however is how in the assignment phase, a cut off point must be set where no more nodes can be assigned to a cluster. If instead we were to give fuzzy values to each of these nodes, we could in turn set up a cut off point based on fuzzy weights. As such, if a node were given weights based on some determined proximity, we could cut off the nodes when we have deemed the cluster to be too large. As such, we can reduce the issue of cutting off nodes from clusters, while also maintaining uniform cluster sizes. I hypothesize that this change could allow for better centroid re estimations as the clusters are now less prone to outliers. Having a cut off number of nodes could result in nodes that are being far away make it into the cluster, and thus cause dilemmas for re estimation. Similarly, this can allow for clusters to contain the same node, as their participation in each cluster is determined by their proximity to a centroid.

Changes To Make Fuzzy Bregman's Clustering Algorithm:

Mainly changes would come from the assignment phase, as this is where fuzzy distance values would be given. Here membership is no longer 0 or 1, but anywhere in between there. Some options lay available however, as we can calculate the distance, the options on how to convert this into a fuzzy weight are many. Below are three proposed solutions:

- Perform the distance from node to centroid as a percentage of the total finite space C. Given node y, Distance(y)/Distance(C). This however may prove difficult to attain the finite space's distance

- We could calculate the fuzzy weight as a percentage of the distance between centroids, as such to represent its percentage in relation to other centroids. Distance(y)/Distance(Centroid). With multiple centroids we could take an average. Unsure of the relevance, and some nodes may be much further than the distance of the centroids are from each other.
- Lastly, we could take the distance from the furthest two nodes, and use that as our denominator. This could prove to be costly and inefficient.

From this point, fuzzy values can be assigned for distance. Now in the Re-estimation phase, instead of limiting ourselves to a set number of nodes, we instead can use a certain fuzzy value as a cut off point. We would need to experiment with different fuzzy values to see what works best, however this would let us set a general size to our cluster in relation to what we are calculating the fuzzy values based off of. Then from this point, re-estimation would occur as it has before, and the whole process would repeat.