This document will cover the various clustering algorithms researched and tested by ECE 18.7. First, Markov Clustering will be covered, as it is the algorithm we chose to continue with, but the other two (group and single linkage) are very promising options and if more research on how to calculate a proper stopping point can be done, they could surpass Markov Clustering. The current implementation of linkage clustering is plug-and-play with HotSpotter, and can be activated by editing two lines of code.

**Introduction**

To identify individual snow leopards in our set of images, we must identify groups of chips that are closely related to each other but show little similarity to other chips. To accomplish this goal, we must implement an algorithm which clusters the chips into discrete sets. Our full set of chips is currently represented by a graph in which the vertices are chips and the edges are scores. This “score graph” is complete (meaning each vertex is connected to every other vertex), and weighted (each edge has a value representing some relationship between two vertices) as shown in Figure 12 below. The clustering algorithm we use should then be able to operate on a graph, preferably a weighted one.

*Figure 10 Example of a score graph*

**Markov Clustering (MCL)**

The currently implemented clustering algorithm works by simulating random walks within a graph by alternating between “expansion” and “inflation” of a matrix M which must follow a set of criteria [3]. M must be stochastic, meaning that each of its columns sums to 1. M is set up such that each entry (*i,j*) is the probability of going from node *j* to node *i* in the random walk. Expansion is matrix squaring (normal matrix product), and inflation is putting a matrix to a power element-by-element, then scaling it so that the resulting matrix is still a stochastic matrix. Eventually this repetition of expansion and inflation yields a matrix in which there are no paths between segments, each segment is then interpreted as a cluster.

We believe a change in clustering will provide the most gains in accuracy, which is the most important aspect of our project. The first problem with using MCL for our score matrix is that MCL is meant to work on a graph whose column sums are all equal to one, but the score matrix created by HotSpotter does not in any way guarantee this, in fact it directly does not follow that convention as each (*i,i*) position in the score matrix is a 1, reflecting that a chip has a 100% match to itself. We are also planning on boosting the scores of chips which are in the same image or are part of the same set of images. This boosting pushes the score matrix even further from the stochastic requirement. The second problem we have noticed with MCL is that it does not cluster tightly enough, which can be changed through adjustment of the inflation\_factor parameter. However, the exact tuning of this parameter has proven difficult, with results varying wildly on different databases.

Because of the shortcomings of MCL, we have researched different clustering algorithms, and have chosen three which we believe to be worth pursuing. We chose these methods because of their existence in already-used Python libraries, compatibility with our dataset, and conceptual simplicity.

**Highly Connected Subgraphs**

The Highly Connected Subgraphs (HCS) method of clustering graphs is more traditional and operates on unweighted graphs [4]. This means that we must first transform our score graph into an unweighted one. To do this, we would threshold the graph, setting all values below a threshold to 0 and all values above the threshold to 1, giving us an unweighted graph. We can then run the HCS algorithm on the new graph.

The HCS algorithm revolves around finding and removing edges which will make a graph disconnected, repeating until each subgraph is highly connected. The minimum cut of a graph is defined as the smallest set of edges that when removed from a graph, result in a graph that is disjoint. A graph on *n* vertices is labeled as being highly connected when the minimum cut of that graph contains more than *n/2* edges. Each highly connected subgraph is then labeled as a cluster, and the HCS algorithm is complete. Figures 13 and 14 below show the original graph and what clusters were identified by HCS respectively.

*Figure 11 Original graph*

*Figure 12 Subgraph clusters identified by HCS*

The main issue with HCS is that the thresholding process reduces the accuracy of the graph as a representation of chip similarity. However, we recently found an algorithm that works for finding minimum cuts on weighted graphs named the Stoer-Wagner algorithm [5]. The Stoer-Wagner algorithm would take into account the scores in our graph as they are, making HCS a much more viable candidate.

**Single Linkage**

Single linkage is an algorithm which falls in the category of hierarchal agglomerative clustering [6]. A hierarchal agglomerative algorithm clusters in a bottom-up fashion, starting with every point in its own cluster and combining clusters until clusters of a large enough size or uniqueness are achieved.

The simple description of single linkage is that at each step it combines two clusters which contain the closest pair of elements that are not yet part of the same cluster. Because of this nature, single linkage can produce clusters which are long and wispy. The effect of such cluster organization is yet to be seen for our dataset. We would have to make some adjustments to existing single linkage algorithms to work with our data. We would need to establish that “distance” between two points is the similarity between two chips. We would also need to find a point at which to stop the clustering, because left unattended it will eventually cluster all points into a single cluster. However, with those two changes in mind, single linkage could prove to be a highly lucrative change for higher accuracy.

It is important to note that the runtime of traditional single linkage is *O(n3)*, though there is an implementation of *O(n2)*. We have been told that accuracy is more important than runtime, so this may not be an issue but should be kept in mind when we compare performance of various clustering algorithms.

**Group Linkage**

Group linkage is, like single linkage, a hierarchal agglomerative clustering method [6]. It operates similarly to single linkage, in that it combines two clusters with the smallest distance. However, in group linkage the two clusters that are combined are the two clusters whose farthest apart elements are the closest. The “distance” between two clusters is defined as the maximum of the distances between each cluster’s elements. The two clusters with the smallest “distance” are then combined.

Group linkage tends to create clusters of more evenly distributed elements (as opposed to single linkage), which could potentially give us better results than single linkage. Like single linkage, we would have to redefine what “distance” meant for our data, as well as determining a point at which to stop clustering. Group linkage also traditionally has a runtime of *O(n3)*, but has faster implementation of *O(n2)*. Once we implement either single linkage or group linkage, the other should be easy to put in place.