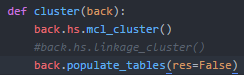
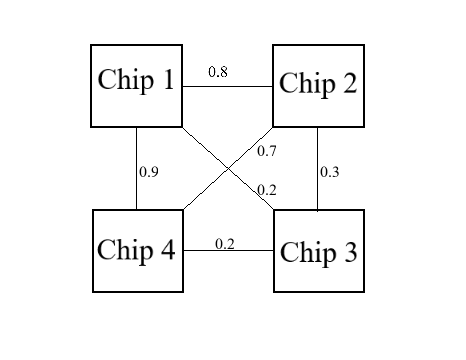
This document will cover the various clustering algorithms researched and tested by ECE 18.7. First, Markov Clustering will be covered, as it is the algorithm we chose to continue with, but the other two (group and single linkage) are very promising options and if more research on how to calculate a 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 in guiback.py, shown below



**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 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 10. The weight of each edge represents the similarity score between two chips, ranging from 0 (no match) to 1 (perfect match). The clustering algorithm we use should then be able to operate on a weighted graph.



*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. M must be stochastic, meaning that each of its columns sums to 1, which is addressed in more detail in the next paragraph. 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.

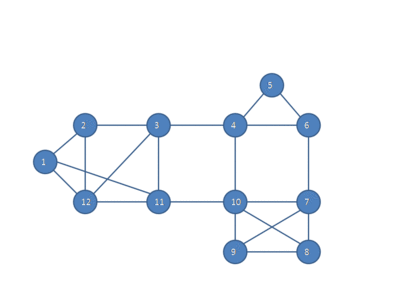
We believed that a change in clustering will provide the most gains in accuracy, which was the most important aspect of our project. The first problem with using MCL for our score matrix is that MCL is meant to work on a graph whose column sums are all equal to one, but the score matrix created by HotSpotter 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 also boosted 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 noticed with MCL implementation is that it does not cluster tightly enough. The MCL implementation has an inflation\_factor parameter that can change how tightly it clusters. However, the exact tuning of this parameter proved difficult, with results varying wildly on different databases.

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

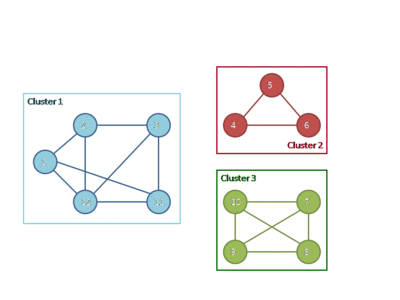
## Highly Connected Subgraphs

The Highly Connected Subgraphs (HCS) method of clustering graphs is more traditional and operates on unweighted graphs. 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 11 and 12 below show an example of an original graph and what clusters were identified by HCS respectively.



*Figure 11 Original graph.*

**

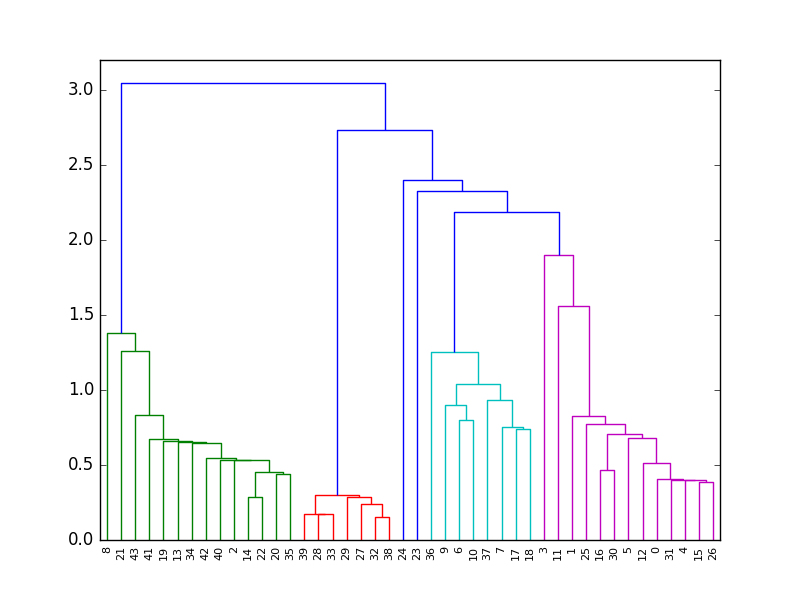
*Figure 12 Subgraph clusters identified by HCS.*

The main issue with HCS is that the thresholding process reduces the accuracy of the graph as a representation of chip similarity. However, we found an algorithm that works for finding minimum cuts on weighted graphs named the Stoer-Wagner algorithm. The Stoer-Wagner algorithm considers the scores in our graph as they are, which allows the HCS algorithm to work well with our dataset and eliminates the need for thresholding. There is an opportunity here for future work, as an algorithm could be written which utilizes the Stoer-Wagner method in conjunction with HCS to separate our score graph into highly connected subgraphs.

## Single Linkage

Single linkage is an algorithm which falls in the category of hierarchal agglomerative clustering . 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, as depicted in Figure 13 below. In Figure 13, the y-axis represents the distance between the two clusters combined, and each vertical like is a cluster. When two clusters are combined, they meet at a horizontal line. The x-axis is the number of the cluster.



*Figure 13 Dendrogram for single linkage*

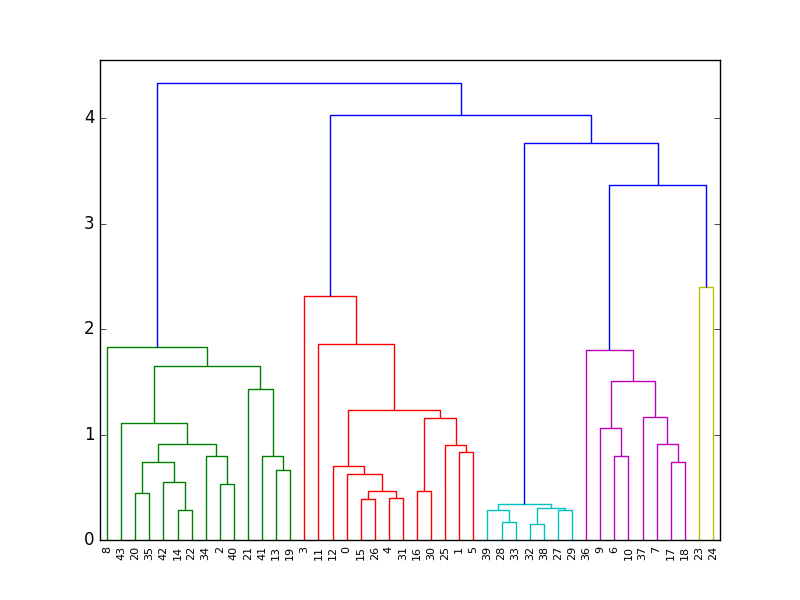
The distance calculated by the linkage clustering method is the similarity score between our chips. Chips that have a higher score are identified as being “closer” by the linkage algorithm. Thus, alike clusters are combined together repeatedly until a single large cluster is formed. Because of this behavior, a stopping point which tells the algorithm when to stop combining clusters is needed. The recommended stopping point always stops the algorithm before it reaches a single large cluster. The downside of this is that linkage clustering will never be able to catch a dataset where all images are of the same cat. Our tests have confirmed that both single and group linkage algorithms are on-par with MCL for some datasets, but additional research into the stopping point is needed to increase both the accuracy and flexibility of the linkage algorithms.

It is important to note that the runtime of traditional single linkage is *O(n3)*, though there is an implementation of *O(n2)*. Accuracy is more important than runtime for the end user, but should still be kept in mind when we compare performance of various clustering algorithms.

## Group Linkage

Group linkage, like single linkage, is a hierarchal agglomerative clustering method. 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 than single linkage, which has provided slightly better results than single linkage. This behavior is shown in Figure 14 below.



*Figure 14 Dendrogram for group 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 either single linkage or group linkage is implemented, the other should be easy to put in place.

## Conclusions

After testing of MCL, single linkage, and group linkage, we decided to stay with MCL, but suggest that if more research is done on a stopping point the linkage clustering methods could outperform MCL. While both single and group linkage were able to equal MCL’s performance in our tests, more research into calculating a stopping point is needed for them to outperform MCL. In our tests, we used a default stopping point, which while providing accurate results for our data is generally unreliable because it would be unable to correctly label a dataset in which all images are of the same cat.

The problems previously mentioned about MCL were fixed by normalizing the scores matrix (this forces the matrix to be stochastic) before passing it to the MCL algorithm. We also identified that an inflation factor of 2 provided the best results across multiple datasets.

HCS in conjunction with the Stoer-Wagner algorithm is another promising candidate that we did not have time to test this year. The Stoer-Wagner allows HCS to run on our dataset without thresholding, which could result in accuracy comparable to MCL.