Clustering Using Graph Connectivity

Patrick Williams

June 3, 2010

1 Introduction

It is often desirable to group elements of a set into disjoint subsets, based on the similarity between the elements in the set. Cluster analysis seeks processes by which this can be done efficiently. Additionally, a good algorithm will hopefully meet two standards: elements within a cluster are highly similar to each other, called *homogeneity*, and elements in separate clusters are dissimilar, called *separation*.

Clustering analysis is desirable in nearly any field of study where it is beneficial to group data into similar sets. Depending on one's objective in analyzing a set of data, one might define similarity between elements differently, and thus a clustering process could be optimized to provide numerous ways of grouping a set of elements.

In order to create any sort of clustering algorithm and determine its effectiveness, it is necessary to find some way to quantify similarity between elements. In this paper this is done by representing the data as an undirected graph, where elements are represented by the vertices of the graph and vertices are connected by an edge if they are "similar". Similarity between elements will usually be judged by the characteristics of the elements themselves.

This paper primarily follows the results of a paper by Hartuv and Shamir[1], which defines the highly connected subgraph and combines this with previous results for finding the minimum cut of a graph to develop an algorithm which computes clusters as highly connected subgraphs in polynomial time.

2 Definitions

A *graph* is defined by two collections (V, E), where V is the set of vertices (sometimes called nodes) in the graph, and E is the set of edges which connect vertices. Thus, two vertices $v, w \in V$ are considered connected if there exists an edge $(v, w) \in E$. A vertex connected by some edge to another vertex is said to be *incident* on that edge. Note that because we are considering undirected graphs, the order of the pair of vertices doesn't matter, i.e. the edges (v, w) and (w, v) are considered equivalent. We also refer to the set of edges and the set of vertices for a graph G by $V(G)$ and $E(G)$, respectively. An *induced subgraph* of a graph G is a subgraph which is given by a subset of the vertices of G, and contains the same edges between these vertices as in the original graph.

A *connected* graph is defined in the intuitive sense, in that a graph is connected if there exists a series of edges (or a path) which connect any two vertices. The *connectivity* of a graph G is denoted by $k(G)$, and is defined by the minimum number of edges which must be removed from the graph in order to create a disconnected graph. A graph with connectivity $l = k(G)$ is said to be l-*connected*. A *clique* is identified as a subset of the vertices of a graph, or the induced subgraph associated with it, where every two vertices are connected by an edge[2]. We will call a graph *highly connected* if $k(G(V,E)) > \frac{|V|}{2}$. The clustering algorithm we will give splits a given graphs into highly connected subgraphs, which are then identified as the clusters we seek.

A *cut* in a graph is a set of edges S which when removed from the graph will disconnect the graph, and a *minimum cut* (or mincut) is a cut which has the fewest possible edges. Therefore, one will notice that any cut S is a minimum cut specifically when $|S| = k(G)$. Finding a minimum cut of a graph is also an important and useful problem in graph theory, and will be discussed later, but isn't covered in quite as much depth in this paper as the clustering algorithm.

Distance $d(u, v)$ between vertices u and v is given by the minimum length (number of edges) of a path between the two vertices, or ∞ if no path exists, and the *diameter* of a connected graph G is given by $\max_{u,v \in V} d(u, v)$. A vertex's *degree*, $deg(v)$, is the number of edges that connect to it. A graph G is said to have a minimum degree $\delta(G) = \min_{v \in V} deg(v)$

Finally, when studying the efficiency of algorithms one of the most important factors being considered is the time taken for it to complete. However, most problems are too complex to say exactly how long an algorithm will take for any given input, so we will usually refer to an algorithm's running time by an upper bound $O(f(n))$. Specifically, we say an algorithm has a bound $O(f(n))$ when there are positive constants c and n_0 such that $T(N) \leq cf(N)$ when $N \geq n_0$. $T(N)$ refers to the "time" it takes to complete the procedure, generally given by the number of constant-time operations (operations which the computer can do in more or less one step, such as adding or checking a logical value) which the algorithm will do in the process of completion. More generally, $O(n)$ notation allows us to provide a limit for how the magnitude of computation grows as the input size n grows large. This understanding of running time for an algorithm is useful in computations where running time becomes a noticable problem for extremely large sets of data. This means we won't really know whether a computation will take 5 minutes or 10 minutes, but it does give us a good idea of whether it will grow to take an hour, a day, a month, or longer when we increase the amount of data by an order of magnitude.

3 Finding a Minimum Cut

The clustering algorithm given by [1] relies on an unspecified minimum cut operation. Several methods for finding the minimum cut of a graph exist. Here we briefly review the minimum cut method for a weighted graph given by Stoer and Wagner[3]. In order to apply this to unweighted graphs, we would simply construct a weighted graph which has an edge weight of 1 wherever the original had an edge. An additional term we will use is the s-t-cut of a graph, which is a cut that specifically separates the two vertices s and t, and thus a minimum s-t-cut is an s-t-cut of minimum weight. *Merging* vertices is an operation where two vertices are replaced by one; the edge from the new vertex to one of the other vertices is given weight equal to the sum of the weight of the edges which connected the original two vertices to the other vertex (0 if no such edge was considered to exist).

This minimum cut algorithm begins with the following theorem:

Theorem 1. *Let s and t be two vertices of a graph G. Let $G/\{s,t\}$ be the graph obtained by merging s and t. Then a minimum cut of G can be obtained by taking the smaller of a minimum s-t-cut of G and a minimum cut of $G/\{s,t\}$.*

The theorem follows from the fact that either the minimum cut separates s and t, and thus is found by the minimum s-t-cut, or the minimum cut doesn't separate s and t, thus the minimum cut of the graph with s and t merged will be the same.

Thus, if we know an algorithm to find an arbitrary minimum s-t-cut of a graph, the theorem lets us recursively find the minimum cut of the graph. Specifically, the algorithm can find a minimum s-t-cut, then merge the two s and t vertices and repeat on the new graph. This can be repeated until the graph is a single vertex where there is no minimum cut, and thus by the theorem the minimum cut must have been the lowest cost cut among the previous minimum s-t-cuts found by the algorithm. The arbitrary minimum s-t-cut algorithm we use is known as the *maximum adjacency search*, which is given by

```
MinimumCutPhase($G, w, a$)
$A \leftarrow \{a\}$
**while** $A \neq V$
    add the mostly tightly connected vertex in $G$ to $A$
store the cut-of-the-phase
shrink $G$ by merging the last two vertices added to $A$
```

The algorithm begins by choosing an arbitrary vertex a to add to the set A, which grows by repetitively grabbing the mostly tightly connected vertex. We choose the most tightly connected vertex as the one for whom the sum of the weights of edges connecting to vertices in A is the maximum.

The cut-of-the-phase is the sum of the weight of all the edges connecting to the last vertex added to A, or equivalently the weight of the cut which separates that vertex. Thus, in this process the minimum cut phase traverses all the vertices of the graph while adding them to A, and takes note of the last two in particular; it bases the cut-of-the-phase on the last, and merges the two.

Stoer and Wagner prove a theorem that each of these cut-of-the-phases found is a minimum s-t-cut of the current graph, where s and t are the two vertices added last in the minimum cut phase[3]. We then obtain the algorithm:

```
MinimumCut($G, w, a$)
while $|V| > 1$
    MinimumCutPhase($G, w, a$)
    if the cut-of-the-phase is less than the current minimum cut
        then store the cut of the phase as the current minimum cut
```

By the two theorems, it's clear that this minimum cut algorithm is correct, as it recursively reduces the graph by merging vertices after it finds their minimum s-t-cut in MinimumCutPhase, ends on a single vertex for which there is no minimum cut. The algorithm yields a running time of $O(|E| + |V|log|V|)$ for MinimumCutPhase, thus since MinimumCut repeats it $|V| - 1$ times (each iteration eliminates one vertex at a time by merging, until only one is left), we have the bound $O(|V||E| + |V|^2 log|V|)$ (note the -1 is irrelevant to the overall $O(n)$ time bound we're describing).

4 The Clustering Algorithm

The clustering algorithm uses a minimum cut operation, this could be the previous algorithm or some other one, but in any case we will assume it provides $(H, \overline{H}, C)$, where the first two sets are the two halves of the cut of the graph G, and C is the set of edges which are removed to make the cut. The Highly Connected Subgraphs algorithm is given as follows:

```
HCS($G(V, E)$)
begin
    $(H, \overline{H}, C) \leftarrow$ MINCUT($G$)
    if $G$ is highly connected
        then return $G$
    else
        HCS($H$)
        HCS($\overline{H}$)
    end if
end
```

Figure 1: The Highly Connected Subgraphs (HCS) algorithm

When the algorithm encounters single vertices which have been split off from a graph they are ignored, or grouped into a separate set of singletons. The collection of subgraphs returned by the algorithm is identified as the clusters of the graph. Note that when we check "G is highly connected" we are specifically checking that $|C| > \frac{|V|}{2}$, since we know that $|C| = k(G)$ for the minimum cut C.

We can easily bound the running time of the algorithm by $O(2N \times f(n, m))$, where N is the number of clusters found and $f(n, m)$ is the time bound for computing MINCUT($G(V, E)$) with $|V| = n$ and $|E| = m$. This is more evident when we notice that unless $f(n, m)$ is in some way reduced to a trivial time bound, it will dominate the time taken for each recursive call of HCS. Therefore, in finding the upper bound of the running time we ignore the time taken in other operations, such as checking if G is highly connected, and consider the number of times MINCUT is called.

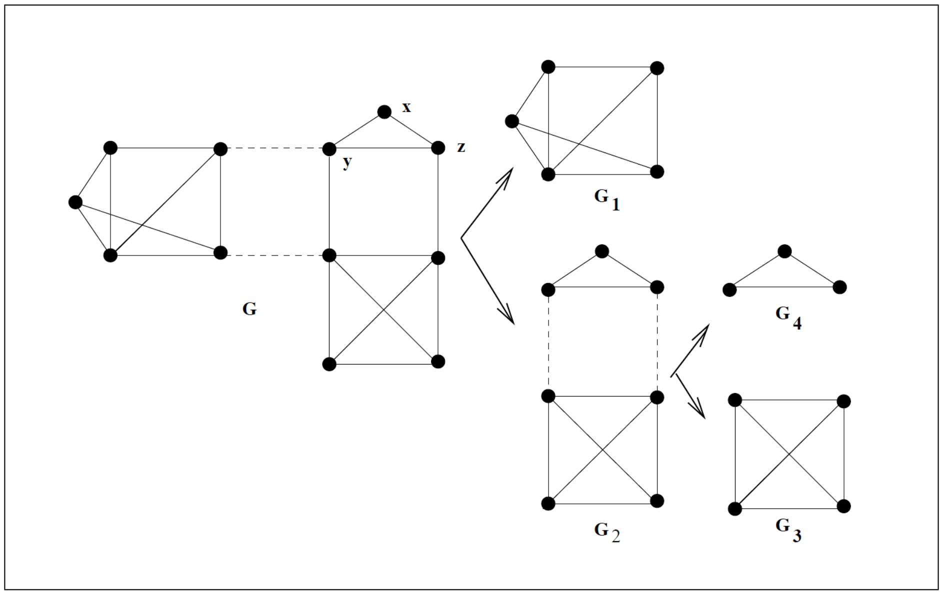

Figure 2: The HCS algorithm on an example graph [1]. Dashed lines indicate the minimum cut used to split the graph. Note other minimum cuts might have been chosen; methods of choosing ideal cuts are discussed later.

To find the maximum number of times the minimum cut will be computed, we consider a full binary tree (a structure where each node has either two nodes or none as children, only one node as a parent, and one root node which has no parent) representing how the graph is split up. At the root node of the tree is the original graph G, and each subsequent MINCUT is represented by the two children of the node where the cut occurs. Therefore, the clusters returned by the algorithm will be the leaves (nodes without children) of the tree, and a minimum cut will have been computed at every node of the tree: at each leaf where the algorithm checks if the subgraph is highly connected, and at the other nodes where it cut the graph and recursively operated on the two parts. It's then fairly clear that the ratio of nodes to leaves will be highest when we have a perfect binary tree (as in all the leaves are at the same depth). A perfect binary tree of depth n will have $2^{n+1} - 1$ nodes and 2^n leaves. Thus, in the worst case this tree will have $2^{n+1} - 1 < 2 * 2^n = 2N$ total nodes, or less than $2N$ calls to MINCUT so we can obtain the bound we wanted to find.

5 Properties of the HCS algorithm

While we've found a relatively efficient time bound for this clustering algorithm, we still wish to show that the clusters given by the algorithm have the similarity traits between elements that we want. In this section we prove several theorems about the highly connected subgraphs the algorithm generates to show their homogeneity and separation.

Theorem 1. *The diameter of every highly connected graph is at most two.*

Proof. If all the edges incident with a vertex v of minimum degree of a graph $G(V, E)$ are removed, then the single vertex will be disconnected from the graph. Therefore, since we can make a disconnection by removing a number of edges equal to the minimum degree of a graph, the edge-connectivity of a graph cannot be greater than its minimum degree ($k(G) \leq \delta(G)$). Adding the restriction that G be highly connected, we obtain $\frac{|V|}{2} < k(G) \leq \delta(G)$. Since the minimum degree is at least half of the number of vertices in G, we have that every vertex is connected by an edge to at least half of the vertices of G. Thus we can conclude that for any $u, v \in V$, $d(u, v) \leq 2$, because they share a common neighbor. More specifically, if they didn't share a common neighbor, that would imply the sets V_1, V_2 of vertices adjacent to u, v would have $V_1 \cap V_2 = \emptyset$. But since $V_1, V_2 > \frac{|V|}{2}$, this would imply that $|V_1 \cup V_2| = |V_1| + |V_2| > |V|$, thus a contradiction. □

It's worth noting that the theorem holds even if we allow $k(G) \geq \frac{|V|}{2}$ as the requirement for a highly connected subgraph, because the vertices counted in $\delta(G)$ don't include the vertex of the minimum degree vertex itself, so the additional two vertices themselves would create the inequality that leads to a contradiction in the previous proof. Additionally, the converse is clearly not true: consider a simple path over three vertices; a cut can be made by removing either one of the edges, which is less than half the number of vertices.

Lemma 1. *If S is a minimum cut which splits the graph into two induced subgraphs, the smaller of which, $\overline{H}$, contains $k > 1$ vertices, then $|S| \leq k$, with equality only if $\overline{H}$ is a clique.*

Proof. We let $deg_G(x)$ denote the degree of vertex x in an induced subgraph G, and $deg_S(x)$ denote the number of edges in the cut S that are incident on x. If we consider $x \in \overline{H}$, we have that $deg_S(x) + deg_{\overline{H}}(x) = deg(x)$. This is evident since all the edges incident on the vertex must connect to either H or $\overline{H}$, and we must have $deg_H(x) = deg_S(x)$ since we assumed $x \in \overline{H}$ can only be connected to vertices in H if the edge is part of the cut being made. Since S is a minimum cut, we can then conclude that

$$deg_S(x) + deg_{\overline{H}}(x) \geq |S|,$$

because otherwise the minimum cut would instead be to remove x as a singleton.

If we use this to consider the sum over all $x \in \overline{H}$, we find

$$\sum_{x \in \overline{H}} deg_S(x) + \sum_{x \in \overline{H}} deg_{\overline{H}}(x) \geq |S||V(\overline{H})|$$

or, since the first sum is of all the edges between H and $\overline{H}$ and the second sum will count the number of edges between vertices in $\overline{H}$ twice, once at each vertex which the edge is incident on, we equivalently obtain

$$|S| + 2|E(\overline{H})| \geq |S||V(\overline{H})|.$$

Subtracting $|S|$,

$$2|E(\overline{H})| \geq |S|(|V(\overline{H})| - 1).$$

By the assumption in the theorem, if $|V(\overline{H})| > 1$, we divide and find

$$|S| \leq \frac{2|E(\overline{H})|}{|V(\overline{H})| - 1}$$

Notice that for any graph, $|E(G)| \leq \frac{|V(G)|(|V(G)|-1)}{2}$. Specifically, this is easy to see if we notice that G is a clique with $|V(G)| > 1$ if and only if the number of edges in G is $|E(G)| = \sum_{n=1}^{|V(G)|-1} n = \frac{|V(G)|(|V(G)|-1)}{2}$ (in the case that $|V(G)| = 1$, the number of edges is clearly zero). This is evident by induction: for a clique of two vertices, there is only one edge between the two, i.e. $\sum_{n=1}^{2-1} n = 1$ and to make a clique of n vertices from a clique of $n-1$ vertices we must add $n-1$ more edges, to connect all the previous vertices to the additional vertex. The other direction is clearly evident by the fact that a graph cannot have the same number of edges and vertices as a clique without being a clique. Thus, we apply this fact to the previous inequality and obtain

$$|S| \leq \frac{2\frac{|V(\overline{H})|(|V(\overline{H})|-1)}{2}}{(|V(\overline{H})| - 1)} = |V(\overline{H})| = k.$$

As we wanted to show, and additionally in the case that the equality holds then we must have the equality $|E(\overline{H})| = \frac{|V(\overline{H})|(|V(\overline{H})|-1)}{2}$ which only occurs if $\overline{H}$ is a clique. □

Notice that this lemma implies that if the minimum cut S of a graph $G = (V, E)$ is such that $|S| > \frac{|V|}{2}$, then the cut only separates a single vertex from the graph, since if $\overline{H}$ is the smaller of the two induced subgraphs, then we must have $|V(\overline{H})| \leq \frac{|V|}{2}$, thus $\frac{|V|}{2} < |S| \leq |V(\overline{H})| \leq \frac{|V|}{2}$ gives us a contradiction, so we must have $k = 1$. This suggests that requiring the connectivity be $\frac{|V|}{2}$ is an good condition to impose, since requiring any higher level of connectivity would only result in singletons being cut off from some graphs.

Theorem 2. *Suppose G is not highly connected but has diameter 2. Let H and $\overline{H}$ be the induced subgraphs obtained by removing a minimum cut S from G,* *where $|V(\overline{H})| \leq |V(H)|$. Then (1) every vertex in $\overline{H}$ is incident on S, (2) $\overline{H}$ is a clique, and (3) if $|V(\overline{H})| > 1$ then every vertex in $\overline{H}$ is incident on a single edge of S.*

Proof. Given that S is a minimum cut and G is not highly connected, then we have two cases:

Case 1, $|S| = \frac{|V|}{2}$: If $\overline{H}$ is a singleton, then this is obviously true, so we consider when $|V(\overline{H})| > 1$. Since we must have $|V(\overline{H})| \leq \frac{|V(G)|}{2}$ and from the previous lemma we must have that $\frac{|V(G)|}{2} = |S| \leq V(\overline{H})$, thus we obtain $V(\overline{H}) = \frac{|V(G)|}{2} = V(H)$, and since both halves are the same size then we can apply the lemma to both to see that both H and $\overline{H}$ must be cliques since the equality holds. Then suppose there exists a vertex $x \in \overline{H}$ which is not incident on an edge in S. Then x is connected by an edge to every other vertex in $\overline{H}$, but none in H, thus $deg(x) = \frac{|V|}{2} - 1 < \frac{|V|}{2}$. But then separating x as a singleton would be a minimum cut, not the original minimum cut, thus we have a contradiction.

Case 2, $|S| < \frac{|V|}{2}$: We first show the strict inequality $|V(\overline{H})| < |V(H)|$ by contradiction. Suppose $|V(\overline{H})| = |V(H)| = \frac{|V|}{2}$. Then since $|S| < \frac{|V|}{2} = |V(\overline{H})| = |V(H)|$, the cut must not have edges adjacent to all the vertices of H or $\overline{H}$, so there must be $u \in H$ and $v \in \overline{H}$ such that u is not adjacent to any vertex in $\overline{H}$ and v is not adjacent to any vertex in H. But then this would imply that $d(u, v) \geq 3$, which contradicts our assumption that the diameter of G is 2.

Thus, we have that $|V(H)| > \frac{|V|}{2} > |S|$. Since there are more vertices in H than edges in S, there must be a vertex $v \in H$ that is not incident on an edge in S. If there were also a vertex $u \in \overline{H}$ that wasn't incident on S, it would imply that $d(u, v) \geq 3$ and thus we have a contradiction as before, so the first assertion (1) is true.

Since every vertex in $\overline{H}$ is incident on S, we have that $|S| \geq |V(\overline{H})|$. When $|V(\overline{H})| = 1$, $\overline{H}$ is clearly a clique. Then suppose $|V(\overline{H})| > 1$. Then by the previous lemma we have $|S| < |V(\overline{H})|$ and thus $|S| = |V(\overline{H})|$. Therefore from the previous lemma we must have that $\overline{H}$ is a clique, and we also must have (3) from $|S| = |V(\overline{H})|$ and (1). □

Theorem 3. *(a) The number of edges in a highly connected subgraph is quadratic with respect to the number of vertices. (b) The number of edges removed by each iteration of the HCS algorithm is at most linear.*

Proof. To obtain (a), let the subgraph $G = (V, E)$ be highly connected, with $|V| = n$. As we showed in the proof of Theorem 1, $\frac{n}{2} < k(G) \leq \delta(G)$ since G is highly connected. Thus, by counting the number of edges where every vertex has minimum degree we obtain the lower bound

$$|E| \geq \frac{1}{2} \times |V| \times \delta(G) > \frac{1}{2} \times n \times \frac{n}{2} = \frac{n^2}{4},$$

where we multiply by $\frac{1}{2}$ because if we count the degree of every vertex we are counting the number of edges twice.

Additionally, we know that for any graph $|E| \leq \frac{|V|(|V|-1)}{2} = \frac{|V|^2-|V|}{2}$. Thus since $|E|$ is bounded both above and below by quadratic terms with respect to the number of vertices, we say that the number of edges of a highly connected subgraph is quadratic with respect to the number of vertices.

To obtain (b), we simply observe that the HCS algorithm splits a graph only when the graph is not highly connected. Thus, the number of edges in the minimum cut made must be less than $\frac{n}{2}$. □

Using these theorems we argue that the HCS algorithm provides suitable clusters. The homogeneity of the clusters generated is evident from theorems 1 and 3(a). Theorem 1 gives us that the clusters returned will have a maximum diameter of two. The only better possibility along these lines would be to require that the diameter be 1, or equivalently that the clusters be cliques, but this requirement would be too strict since any false negative errors when determining similarity between elements would cause an element to be thrown out. From Theorem 3(a) we obtain that the clusters are at least half as dense as a clique, which makes sense since the minimum degree is at least half that of a clique.

The separation of the clusters is also evident from Theorem 3. From (a) we see that the number of edges in clusters identified by the algorithm will be quadratic, whereas (b) shows that the number of edges in potential clusters that are "rejected" by the algorithm and split again is linear with respect to the size of the clusters. Thus, unless the clusters are relatively small, there is a significant difference between the number of edges connecting subgraphs that are considered clusters and those that are not.

Theorem 2 also provides a good provision for the separation of the algorithm in that subgraphs with a diameter of two are not likely to be split by the algorithm except in cases which are trivial or which would occur rarely with true clusters affected by noise. Suppose we have some subgraph $G\prime$ with diameter less than or equal to two, which the algorithm splits into two sets of vertices C_1 and C_2 by a cut S, with C_1 being the smaller of the two sets. Then if $|C_1| > 1$ by Theorem 3 every vertex of C_1 is incident to a single edge in the cut, and thus adjacent to only a single vertex of C_2, and not to any other vertices in C_2, and additionally C_2 is a clique. Thus, if the whole subgraph $G\prime$ were really a cluster with missing edges due to noise in the data, this noise would have caused the number of edges between C_1 and C_2 to be $|C_1|$. Without noise we would expect this number to grow more like $|C_1|^2$ with respect to the number of vertices, since there are at least $|C_1|$ vertices in C_2 which would likely be connected with the vertices of C_1 if $G\prime$ were a true cluster of nontrivial size. Additionally, this discrepancy in the connectivity of $G\prime$ is of a rather particular structure, since these missing edges cause such a mincut to be formed, while C_1 is entirely a clique. Thus, it's rather unlikely that random noise would cause the algorithm to make this split when a subgraph should be identified as a cluster. From this we can further conclude that the algorithm will usually only split sets with a diameter of at least three, which shows that the algorithm creates a reasonable and strong boundary line between what is considered a cluster and what is not.

6 Improvements

Hartuv and Shamir also describe some heuristic improvements to the algorithm which improve the performance of the algorithm and the clusters it produces.

6.1 Iteration

If a graph has several possible minimum cuts, the one which is most beneficial for clustering might not be chosen. For example, at for any of the cuts we could have chosen the edges (x, y) and (x, z) instead, which would have split x off as a singleton, and resulted in y and z being split off as singletons as well, in many similar cases these choices result in clusters being broken into singletons. To solve this problem, it's possible to simply run the algorithm again multiple times, initially removing the clusters which were previously found, until new clusters are not found. Theoretically this could cause the running time to be longer, though for an average set of data the number of iterations necessary is usually relatively few.

6.2 Singletons adoption

They also suggest allowing clusters to adopt singletons by checking whether the singletons have more neighbors in the clusters than in the set of singletons. This is done several times, similarly to how we repeat the HCS algorithm, since depending on the order the singletons are processed to be adopted the number of neighbors they have in different clusters might change as they're adopted.

6.3 Initially Removing Low Degree Vertices

When the graph contains low degree vertices, the minimum cut of the graph will often be to simply remove these vertices as singletons. Since computing the minimum cut of the graph is a time consuming process, and may happen often, we can significantly shorten the running time of the algorithm by allowing it to remove low-degree vertices before attempting the HCS algorithm. We end up with the improved algorithm in Figure 3.

The accuracy of this and other clustering algorithms were compared with a simple evaluation. For a graph of n vertices, a clustering solution is represented by an $n \times n$ matrix, M where $M_{ij} = 1$ where vertex i and j are in the same cluster, and 0 where they are not. Then given a true solution matrix T, the accuracy of M is measured by the *Minkowski score* $||T - M||/||T||$. Thus, the smaller the score the better. Tested on both simulated and real data, the HCS algorithm performed well in comparison to other clustering algorithms.

```
HCS_LOOP($G(V, E)$)
begin
    for (i=1 to p) do
        remove clustered vertices from $G$
        $H \leftarrow G$
        repeatedly remove all vertices of degree $< d_i$ from $H$
        until (no new cluster is found by HCS) do
            HCS($H$)
            perform singletons adoption
            remove identified clusters from $H$
        end until($\overline{H}$)
    end for
end
```

Figure 3: The HCS algorithm with heuristic improvements

7 Conclusion

Hopefully this has provided insight into both the HCS algorithm and the problem of clustering sets of data as well. It should be clear how this process of analysis can be widely applied in any field.

A detail which was somewhat glossed over in the beginning was how the similarity graph is generated from the data. Depending on the level of noise in the data, choosing an appropriate threshold to define elements as ”similar” lead to errors in grouping, or could result in very different groupings depending on the stringency of the threshold. While this also provides some room for variation in interpreting data it is desirable to find a way to determine the best threshold necessary to ensure ”good” clustering given some noise level.

There is plenty of room for improvement to this algorithm; besides improvement in speed it could be generalized to work for weighted graphs, where each edge is given a weight that represents the degree of similarity between two elements. While the method for finding the minimum cut that we found works for weighted graphs, it is difficult to appropriately define a ”highly connected subgraph” for a weighted graph; we cannot simply count the number or weight of edges, since this won't necessarily imply that the weight of edges contained in the graph are ”evenly spread” throughout the vertices. One vertex might simply have all the edges incident with it, or have the highest weight edges. The simplest solution would be to choose a threshold for which items would be considered similar or not, and generate an unweighted graph to use the HCS algorithm on. This would probably provide satisfactory results, but it again highlights the question of how to quantify similarity in a set of data without a strict definition. Especially in a weighted graph, clusters are difficult to define because similarity is vaguely defined. One approach to this might be based on a statistical analysis, where the clusters are identified by ensuring the edge weights are within some tolerance of similarity. This sort of approach could provide a much better way of analyzing a set of data, because instead of trying to define a threshold of similarity based on the data, the threshold could be varied in order to find broad mildly similar groups which are narrowed down to highly similar sub-clusters. Depending on the complexity of the statistical analysis necessary, it is of interest to look into the practicality of such an approach.

References

[1] Erez Hartuv and Ron Shamir. A clustering algorithm based on graph connectivity. *Information Processing Letters*, 76(4-6):175–181, December 2000.

[2] Michael Sipser. *Introduction to the Theory of Computation.* Course Technology, Cengage Learning, Boston, Massachusetts, 2nd edition, 2006.

[3] Mechthild Stoer and Frank Wagner. A simple min cut algorithm. *Journal of the ACM*, 44(4):585–591, July 1997.