Graph Clustering and Minimum Cut Trees

Gary William Flake[1] Robert E. Tarjan[2] Kostas Tsioutsiouliklis[3]

[1] Yahoo! Research Labs / 74 North Pasadena Ave. 3rd floor / Pasadena, CA 91103
flake@yahoo-inc.com

[2]Computer Science Department / Princeton University / 35 Olden Street / Princeton, NJ 08544
ret@cs.princeton.edu

[3]NEC Laboratories America / 4 Independence Way / Princeton, NJ 08540
kt@nec-labs.com

April 1, 2004

Abstract

In this paper, we introduce simple graph clustering methods based on minimum cuts within the graph. The clustering methods are general enough to apply to any kind of graph but are well-suited for graphs where the link structure implies a notion of reference, similarity, or endorsement, such as Web and citation graphs. We show that the quality of the produced clusters is bounded by strong minimum cut and expansion criteria. We also develop a framework for hierarchical clustering and present applications to real-world data. We conclude that the clustering algorithms satisfy strong theoretical criteria and perform well in practice.

1 Introduction

Clustering data sets into disjoint groups is a problem arising in many domains. From a general point of view, the goal of clustering is to find groups that are both homogeneous and well separated, that is, entities within the same group should be similar and entities in different groups dissimilar. Often, data sets can be represented as weighted graphs, where nodes correspond to the entities to be clustered and edges correspond to a similarity measure between those entities.

The problem of graph clustering is well studied and the literature on the subject is very rich [5, 15, 16]. The best known graph clustering algorithms attempt to optimize specific criteria such as k-median, minimum sum, minimum diameter, etc. [1]. Other algorithms are application-specific and take advantage of the underlying structure or other known characteristics of the data. Examples are clustering algorithms for images [24] and program modules [19].

In this paper, we present a new clustering algorithm which is based on maximum flow techniques, and in particular minimum cut trees. Maximum flow algorithms are relatively fast and simple, and have been used in the past for data-clustering (e.g. [24], [6]). The main idea behind maximum flow (or equivalently, minimum cut [7]) clustering techniques is to create clusters that have small inter-cluster cuts (i.e. between clusters) and relatively large intra-cluster cuts (i.e. within clusters). This guarantees strong connectedness within the clusters and is also a strong criterion for a good clustering, in general.

Our clustering algorithm is based on inserting an artificial sink into a graph that gets connected to all nodes in the network. Maximum flows are then calculated between all nodes of the network and the artificial sink. A similar approach has been introduced in [6], which was also the motivation for our work. In this paper we analyze the minimum cuts produced, calculate the quality of the clusters produced in terms of expansion-like criteria, generalize the clustering algorithm into a hierarchical clustering technique, and apply it to real-world data.

This paper consists of six sections. In Section 2, we review basic definitions necessary for the rest of the paper. In Section 3, we focus on the previous work of Flake *et al.* [6], and then describe our cut clustering

algorithm and analyze it in terms of upper and lower bounds. In Section 4, we generalize the method of Section 3 by defining a hierarchical clustering method. In Section 5, we present results of our method applied to three real-world problem sets and see how the algorithm performs in practice. The paper ends with Section 6, which contains a summary of our results and final remarks.

2 Basic Definitions and Notation

2.1 Minimum Cut Tree

Throughout this paper, we represent the Web (or a subset of the Web) as a directed graph $G(V, E)$, with $|V| = n$ vertices, and $|E| = m$ edges. Each vertex corresponds to a Web page, and each directed edge to a hyperlink. The graph may be weighted, in which case we associate a real valued function w with an edge, and say, for example, that edge $e = (u, v)$ has weight (or capacity) $w(e) = w(u, v)$. Also, we assume that G is connected; if that is not the case one can work on each connected component independently and the results still apply.

Two subsets A and B of V, such that $A \cup B = V$ and $A \cap B = \emptyset$, define a cut in G, which we denote as (A, B). The sum of the weights of the edges crossing the cut defines its value. We use a real function c to represent the value of a cut. So, cut (A, B) has value $c(A, B)$. In fact, function c can also be applied when A and B do not cover V, but $A \cup B \subset V$.

Our algorithms are based on minimum cut trees, which were defined in [13]. For graph G, there exists a weighted graph T_G, which we call the *minimum cut tree* (or simply, *min-cut tree*) of G. The min-cut tree is defined over V and has the property that we can find the minimum cut between two nodes s, t in G by inspecting the path that connects s and t in T_G. The edge of minimum capacity on that path corresponds to the minimum cut. The capacity of the edge is equal to the minimum cut value, and its removal yields two sets of nodes in T_G, but also in G, which correspond to the two sides of the cut. For every undirected graph, there always exists a min-cut tree. Gomory and Hu [13] describe min-cut trees in more detail and provide an algorithm for calculating min-cut trees.

2.2 Expansion and Conductance

In this subsection, we discuss a small number of clustering criteria and compare and contrast them to one another. We initially focus on two related measures of intra-cluster quality and then briefly relate these measures to inter-cluster quality.

Let $(S, \bar{S})$ be a cut in G. We define the expansion of a cut as [16]:

$$\psi(S) = \frac{\sum_{u \in S, v \in \bar{S}} w(u, v)}{\min\{|S|, |\bar{S}|\}},$$

where $w(u, v)$ is the weight of edge (u, v). The expansion of a (sub)graph is the minimum expansion over all the cuts of the (sub)graph. The quality of a clustering can be measured in terms of expansion: The expansion of a clustering of G is the minimum expansion over all its clusters. The larger the expansion of the clustering the higher its quality. One way to think about the expansion of a clustering is that it serves as a strict lower-bound for the expansion of any intra-cluster cut for a graph.

We can also define conductance, which is similar to expansion except that it weights cuts inversely by a function of edge weight instead of the number of vertices in a cut set. For the cut $(S, \bar{S})$ in G, conductance is defined as [16]:

$$\phi(S) = \frac{\sum_{u \in S, v \in \bar{S}} w(u, v)}{\min\{c(S), c(\bar{S})\}},$$

where $c(S) = c(S, V) = \sum_{u \in S} \sum_{v \in V} w(u, v)$. As with expansion, the conductance of a graph is the minimum conductance over all the cuts of the graph. For a clustering of G, let $C \subseteq V$ be a cluster and $(S, C - S)$ a cluster within C, where $S \subseteq C$. The conductance of S in C is [16]:

$$\phi(S, C) = \frac{\sum_{u \in S, v \in C - S} w(u, v)}{\min\{c(S), c(C - S)\}}.$$

The conductance of a cluster $\phi(C)$ is the smallest conductance of a cut within the cluster; for a clustering the conductance is the minimum conductance of its clusters. Thus, the conductance of a clustering acts as a strict lower-bound on the conductance of any intra-cluster cut.

Both expansion and conductance seem to give very good measures of quality for clusterings and are claimed in [16] to be generally better than simple minimum cuts. The main difference between expansion and conductance is that expansion treats all nodes as equally important while conductance gives greater importance to nodes with high degree and adjacent edge weight.

However, both expansion and conductance are insufficient by themselves as clustering criteria because neither enforces qualities pertaining to inter-cluster weight, nor the relative size of clusters. Moreover, there are difficulties in directly optimizing either expansion or conductance because both are computationally hard to calculate, usually raising problems that are NP-hard, since they each have an immediate relation to the *sparsest cut* problem. Hence, approximations must be employed.

With respect to the issue of simultaneously optimizing both intra- and inter- cluster criteria, Kannan *et al.* [16] have proposed a bicriteria optimization problem that requires:

1. Clusters must have some minimum conductance (or expansion) α, and
2. The sum of the edge weights between clusters must not exceed some maximum fraction ϵ of the sum of the weights of all edges in G.

The algorithm that we are going to present has a similar bicriterion. It provides a lower bound on the expansion of the produced clusters and an upper bound on the edge weights between each cluster and the rest of the graph.

3 Cut Clustering Algorithm

3.1 Background and Related Work

In this section we present our basic clustering algorithm, which we call *cut clustering algorithm*. The analysis of the cut clustering algorithm is based on minimum cut trees. While flows and cuts are well-defined for both directed and undirected graphs, minimum cut trees are defined only for undirected graphs. Therefore, we restrict, for now, the domain to undirected graphs. In Section 5 we will address the issue of edge direction and how we deal with it.

Let $G(V, E)$ be an undirected graph, and let $s, t \in V$ be two nodes of G. Let (S, T) be the minimum cut between s and t, where $s \in S$ and $t \in T$. We define S to be the *community* of s in G with respect to t. If the minimum cut between s and t is not unique, we choose the minimum cut that minimizes the size of S. In that case, it can be shown that S is unique [13].

The work presented in this paper was motivated by the paper of Flake *et al.* [6]. There, a *Web community* is defined in terms of connectivity. In particular, a Web community S is defined as a collection of nodes that has the property that all nodes of the Web community predominantly link to other Web community nodes. That is:

$$\sum_{v \in S} w(u, v) > \sum_{v \in \bar{S}} w(u, v), \forall u \in S \tag{1}$$

We briefly establish a connection between our definition of a community and Web communities from [6]. The following lemma applies:

Lemma 3.1 *For an undirected graph $G(V, E)$, let S be a community of s with respect to t. Then,*

$$\sum_{v \in S} w(u, v) > \sum_{v \in \bar{S}} w(u, v), \forall u \in S - \{s\} \tag{2}$$

Proof. Since S is a community of s with respect to t, the cut $(S, V - S)$ is a minimum cut for s and t in G. Thus, every node $w \in S - \{s\}$ has more edge weight to the nodes of S than to the nodes of $V - S$; otherwise, moving w to the other side of the cut (S, T) would yield a smaller cut between s and t. ■

The above lemma shows that a community S based on minimum cuts is also a Web community. The only node to possibly violate the Web community property is the source s itself. In that case, it is usually a good indication that the community of s should actually be a larger community that contains S as a subset. At this point we should also point out the inherent difficulty of finding Web communities with the following theorem:

Theorem 3.2 *Let $G(V,E)$ be a graph and $k > 1$ an integer. The problem of partitioning G into k Web communities is NP-complete.*

Proof. The proof is given in the appendix. (Note that k can be at most $n/2$, since no single node can form a community by itself. Also, $k > 1$, since for $k = 1$, the set of all nodes forms a trivial community.) ■

3.2 Cut Clustering Algorithm

At this point we extend the definition of a community S of s, when no t is given. To do this, we introduce an artificial node t, which we call the *artificial sink*. The artificial sink is connected to all nodes of G via an undirected edge of capacity α. We can now consider the community S of s as defined before, but with respect to the artificial sink, t. We can now show our main theorem:

Theorem 3.3 *Let $G(V,E)$ be an undirected graph, $s \in V$ a source, and connect an artificial sink t with edges of capacity α to all nodes. Let S be the community of s with respect to t. For any non-empty P and Q, such that $P \cup Q = S$ and $P \cap Q = \emptyset$, the following bounds always hold:*

$$\frac{c(S, V-S)}{|V-S|} \leq \alpha \leq \frac{c(P,Q)}{\min(|P|,|Q|)} \tag{3}$$

Theorem 3.3 shows that α serves as an upper-bound of the inter-community edge capacity and a lower bound of the intra-community edge-capacity. In Section 2.2 we saw that the expansion for a subset S of V is defined as the minimum value of $\phi(S) = \frac{c(P,Q)}{\min(|P|,|Q|)}$, for all cuts (P,Q) of S. Thus, a community of G with respect to the artificial sink will have expansion at least α. The left side of inequality 3 bounds the inter-community edge-capacity, thus guaranteeing that communities will be relatively disconnected. Thus, α can be used to control the trade-off between the two criteria. We will discuss the choice of α and how it affects the community more extensively in Subsection 3.3.

To prove Theorem 3.3 we prove a series of lemmata. But first we define G_α to be the *expanded graph* of G, after the artificial sink t is connected to all nodes V with weight α.

Lemma 3.4 *Let $s, t \in V$ be two nodes of G and let S be the community of s with respect to t. Then, there exists a min-cut tree T_G of G, and an edge $(a,b) \in T_G$, such that the removal of (a,b) yields S and $V - S$.*

Note that the issue raised in this lemma is that there may be multiple min-cut trees for G. Despite this fact, there always exists at least one min-cut tree T_G, for which the removal of an edge of T_G yields exactly S and $V - S$.

Proof. Gomory and Hu prove the existence of a min-cut tree for any undirected graph G by construction. Their proof starts off with an arbitrary pair of nodes a, b, finds a minimum cut (A,B) between a and b, and constructs the min-cut tree recursively. The resulting min-cut tree contains (a,b) as an edge. To prove the lemma, it suffices to pick $a = s$, $b = t$, and the minimum cut $(S, V-S)$ as starting points. ■

Lemma 3.5 *Let T_G be a min-cut tree of a graph $G(V,E)$, and let (u,w) be an edge of T_G. Edge (u,w) yields the cut (U,W) in G, with $u \in U$, $w \in W$. Now, take any cut (U_1,U_2) of U, so that U_1 and U_2 are non-empty, $u \in U_1$, $U_1 \cup U_2 = U$, and $U_1 \cap U_2 = \emptyset$. Then:*

$$c(W,U_2) \leq c(U_1,U_2) \tag{4}$$

Note that this lemma defines a type of triangle inequality for flows and cuts that will be frequently used in the proofs that follow.

Proof. Since (u, w) is an edge of T_G, it defines a minimum cut (U, W) between u and w in G. Now consider the cut $(U_1, W \cup U_2)$. Since $u \in U_1$ and $w \in W$, the cut $(U_1, W \cup U_2)$ is also a cut between u and w, but not necessarily minimum. Hence, we can write:

$$c(U, W) \leq c(U_1, W \cup U_2) \tag{5}$$
$$c(U_1 \cup U_2, W) \leq c(U_1, W \cup U_2) \tag{6}$$
$$c(U_1, W) + c(U_2, W) \leq c(U_1, W) + c(U_1, U_2) \tag{7}$$
$$c(U_2, W) \leq c(U_1, U_2) \tag{8}$$

■

Lemma 3.6 *Let G_α be the expanded graph of G, and let S be the community of s with respect to the artificial sink t. For any non-empty P and Q, such that $P \cup Q = S$ and $P \cap Q = \emptyset$, the following bound always holds:*

$$\alpha \leq \frac{c(P, Q)}{\min(|P|, |Q|)} \tag{9}$$

Proof. Let T_{G_α} be a minimum cut tree of G_α, and let (s', t') be an edge of T_{G_α} whose removal yields S and $V - S \cup \{t\}$ in G_α. According to Lemma 3.4, such a T_{G_α} exists, and by the definition of a min-cut tree, (s', t') lies on the path from s to t in T_{G_α}. (See Figure 1.)

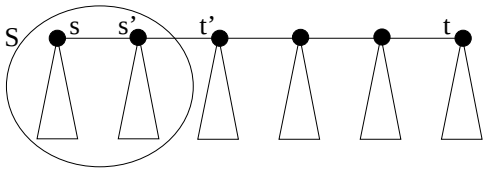

Figure 1: Min-cut tree T_{G_α} with emphasis on the (s,t)-path. Triangles correspond to subtrees that are not part of the path.

Now, we split S into two subsets P and Q, and assume w.l.o.g. that $s' \in P$. Then, Lemma 3.5 applies and yields:

$$c(V - S \cup \{t\}, Q) \leq c(P, Q) \tag{10}$$
$$c(V - S, Q) + c(\{t\}, Q) \leq c(P, Q) \tag{11}$$
$$c(\{t\}, Q) \leq c(P, Q) \tag{12}$$
$$\alpha |Q| \leq c(P, Q) \tag{13}$$
$$\alpha \cdot \min(|P|, |Q|) \leq c(P, Q) \tag{14}$$

Inequality 13 is true because t is connected to every node in V with an edge of weight α, and inequality 14 implies the lemma. ■

Lemma 3.7 *Let G_α be the expanded graph of $G(V, E)$ and let S be the community of s with respect to the artificial sink t. Then, the following bound always holds:*

$$\frac{c(S, V - S)}{|V - S|} \leq \alpha \tag{15}$$

Proof. As in the previous lemma, let T_{G_α} be a minimum cut tree of G_α that contains an edge (s', t'), the removal of which yields sets S and $V - S \cup \{t\}$. By the definition of a min-cut tree, (s', t') corresponds to a minimum cut between s' and t' in G_α. The value of this minimum cut is equal to $c(S, V - S \cup \{t\})$.

Now, consider the cut $(V, \{t\})$ in G_α. It also separates s from t, but is not necessarily a minimum cut between them. This means:

$$\begin{aligned} c(S, V - S \cup \{t\}) &\leq c(V, \{t\}) && (16)\\ c(S, V - S) + c(S, \{t\}) &\leq c(V - S, \{t\}) + c(S, \{t\}) && (17)\\ c(S, V - S) &\leq c(V - S, \{t\}) && (18)\\ c(S, V - S) &\leq \alpha |V - S| && (19) \end{aligned}$$

The last inequality implies the lemma and follows from the fact that t is connected to all nodes of the graph by an edge of weight α. ■

Now we can prove **Theorem 3.3**:
Proof. Immediate by combination of Lemmata 3.6 and 3.7. ■

We can use the above ideas to either find high-quality communities in G, or to develop a general clustering algorithm based on the min-cut tree of G. Figure 2 shows the cut clustering algorithm. The idea is to expand G to G_α, find the min-cut tree T_{G_α}, remove the artificial sink t from the min-cut tree, and the resulting connected components form the clusters of G.

```
CutClustering_Algorithm(G(V, E), α)
    Let V' = V ∪ t
    For all nodes v ∈ V
        Connect t to v with edge of weight α
    Let G'(V', E') be the expanded graph after connecting t to V
    Calculate the minimum-cut tree T' of G'
    Remove t from T'
    Return all connected components as the clusters of G
```

Figure 2: Cut clustering algorithm.

So, Theorem 3.3 applies to all clusters produced by the cut clustering algorithm, giving lower bounds on their expansion and upper bounds on the inter-cluster connectivity.

3.3 Choosing α

We have seen that the value of α in the expanded graph G_α plays a crucial role in the quality of the produced clusters. But how does it relate to the number and sizes of clusters, and what is a good choice for α?

It is easy to see that as α goes to 0, the min-cut between t and any other node v of G will be the trivial cut $(\{t\}, V)$, which isolates t from the rest of the nodes. So, the cut clustering algorithm will produce only one cluster, namely the entire graph G, as long as G is connected. On the other extreme, as α goes to infinity it will cause T_{G_α} to become a star with the artificial sink at its center. Thus, the clustering algorithm will produce n trivial clusters, all singletons.

For values of α between these two extremes the number of clusters will be between 1 and n, but the exact value depends on the structure of G and the distribution of the weights over the edges. What is important, though, is that as α increases the number of clusters is non-decreasing. When implementing our algorithm we often need to apply a binary-search-like approach in order to determine the best value for α, or we can make use of the nesting property of min-cuts.

The nesting property of min-cuts has been used by Gallo *et al.* [8] in the context of parametric maximum flow algorithms. A parametric graph is defined as a regular graph G with source s and sink t, with edge weights being linear functions of a parameter λ, as follows

1. $w_\lambda(s, v)$ is a non-decreasing function of λ for all $v \neq t$.
2. $w_\lambda(v, t)$ is a non-increasing function of λ for all $v \neq s$.

3. $w_\lambda(v, w)$ is constant for all $v \neq s$, $w \neq t$.

A maximum flow (or minimum cut) in the parametric graph G corresponds to a maximum flow (or minimum cut) in G for some particular value of λ. One can immediately see that the parametric graph is a generalized version of our expanded graph G_α, since the edges adjacent to the artificial sink are a linear function of α, and no other edges in G_α are parametric. We can use both non-decreasing and non-increasing values for α, since t can be treated as either the sink or the source in G_α.

The following nesting property lemma holds for G_α:

Lemma 3.8 *For a source s in G_{α_i}, where $\alpha_i \in \{\alpha_1, ..., \alpha_{max}\}$, such that $\alpha_1 < \alpha_2 < ... < \alpha_{max}$, the communities $S_1, ..., S_{max}$ are such that $S_1 \subseteq S_2 \subseteq ... \subseteq S_{max}$, where S_i is the community of s with respect to t in G_{a_i}.*

Proof. This is a direct result of a similar lemma in [8]. ■

In fact, in [8] it has been shown that for a given s the total number of different communities S_i is no more than $n - 2$ regardless of the number of a_i's and they can all be computed in time proportional to a single max-flow computation, by using a variation of the Goldberg-Tarjan preflow-push algorithm [11].

Thus, if we want to find a cluster of s in G of certain size or other characteristic we can simply use this methodology, find all clusters fast, and then choose best clustering according to a metric of our choice. Also, because the parametric preflow algorithm in [8] finds all clusters either in increasing or decreasing order, we can stop the algorithm as soon as a desired cluster has been found. We will use the nesting property again, in Section 4, when describing the hierarchical cut clustering algorithm.

3.4 Heuristic for the Cut Clustering Algorithm

The running time of the basic cut clustering algorithm is equal to the time to calculate the minimum cut tree, plus a small overhead for extracting the subtrees under t. But calculating the min-cut tree can be equivalent to computing $n - 1$ maximum flows [13], in the worst case. Therefore, we use a heuristic that, in practice, finds the clusters of G much faster, usually in time proportional to the total number of clusters.

Lemma 3.9 *Let $v_1, v_2 \in V$ and S_1, S_2 be their communities with respect to t in G_α. Then either S_1 and S_2 are disjoint or the one is a subset of the other.*

Proof. This is a special case of a more general lemma in [13]. If S_1 and S_2 overlap without the one containing the other, then either $S_1 \cap S_2$ or $S_1 - S_2$ is a smaller community for v_1, or symmetrically, either $S_1 \cap S_2$ or $S_2 - S_1$ is a smaller community for v_2. Thus S_1 and S_2 are disjoint or the one is a subset of the other. ■

A closer look at the cut clustering algorithm shows that it suffices to find the minimum cuts that correspond to edges adjacent to t in T_{G_α}. So, instead of calculating the entire min-cut tree of G_α, we use Lemma 3.9 in order to reduce the number of minimum cut computations necessary. If the cut between some node v and t yields the community S, then we do not use any of the nodes in S as subsequent sources to find minimum cuts with t, since according to the previous lemma they cannot produce larger communities. Instead, we mark them as being in community S, and later if S becomes part of a larger community S' we mark all nodes of S as being part of S'. Obviously, the cut clustering algorithm finds the largest communities between t and all nodes in V.

The heuristic relies on the choice of the next node for which we find its minimum cut with respect to t. The larger the next cluster, the smaller the number of minimum cut calculations. Prior to any min-cut calculations, we sort all nodes according to the sum of the weights of their adjacent edges, in decreasing order. Each time, we compute the minimum cut between the next unmarked node and t. We have seen, in practice, that this reduces the number of max-flow computations almost to the number of clusters in G, speeding up the algorithm significantly.

4 Hierarchical Cut Clustering Algorithm

We have seen that the cut clustering algorithm produces a clustering of the nodes of G for a certain value α. Once such a clustering is produced, we can contract the clusters into single nodes and apply the same algorithm to the resulting graph. When contracting a set of nodes, they get replaced by a single new node; possible loops get deleted and parallel edges are combined into a single edge with weight equal to the sum of their weights.

When applying the cut clustering algorithm on the contracted graph, we simply choose a new α-value, and the new clustering, now, corresponds to a clustering of the clusters of G. By repeating the same process a number of times, we can build a hierarchy of clusters. Notice that the new α has to be of smaller value than the last, for otherwise the new clustering will be the same, since all contracted nodes will form singleton clusters.

The quality of the clusters at each level of the hierarchy is the same as for the initial basic algorithm, depending each time on the value of α, only the expansion measure is now over the clusters instead over the nodes. The iterative cut clustering algorithm stops either when the clusters are of a desired number and/or size, or when only a single cluster that contains all nodes is identified. The hierarchical cut clustering algorithm is described in Figure 3.

```
HIERARCHICAL_CUTCLUSTERING(G(V, E))
    Let G^0 = G
    For (i = 0; ; i++)
        Set new, smaller value α_i /* possibly parametric */
        Call CutCluster_Basic(G^i, α_i)
        If ((clusters returned are of desired number and size) or
            (clustering failed to create non-trivial clusters))
                break
        Contract clusters to produce G^{i+1}
    Return all clusters at all levels
```

Figure 3: Hierarchical cut clustering algorithm.

The hierarchical cut clustering algorithm provides a means to look at graph G in a more structured, multi-level way. We have used this algorithm in our experiments to find clusters at different levels of locality. Figure 4 shows how a hierarchical tree of clusters looks like. At the lower levels α is large and the clusters are small and dense. At higher levels, the clusters are larger in size and sparser. Also notice that the clusters at higher levels are always supersets of clusters at lower levels. This is expressed in the following lemma, the proof of which follows directly from the nesting property:

Lemma 4.1 *Let $\alpha_1 > \alpha_2 > ... > \alpha_{max}$ be a sequence of parameter values that connect t to V in G_{a_i}. Let $\alpha_{max+1} \leq \alpha_{max}$ be small enough to yield a single cluster in G and $\alpha_0 \geq \alpha_1$ be large enough to yield all singletons. Then all α_{i+1} values, for $0 \leq i \leq max$, yield clusters in G which are supersets of the clusters produced by each α_i, and all clusterings together form a hierarchical tree over the clusterings of G.*

5 Experimental Results

5.1 CiteSeer

CiteSeer [25], is a digital library for scientific literature. Scientific literature can be viewed as a graph, where the documents correspond to the nodes and the citations between documents to the edges that connect them. The cut clustering algorithms require the graph to be undirected, but citations, like hyperlinks, are directed. In order to transform the directed graph into undirected, we normalize over all out-bound edges for each node (so that the total sums to unity), remove edge-directions, and combine parallel edges. Parallel edges resulting from two directed edges are resolved by summing the combined weight.

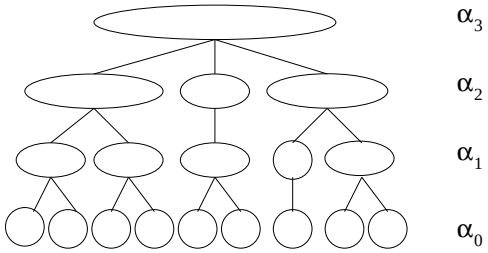

Figure 4: Hierarchical tree of clusters.

Our normalization is similar to the first iteration of HITS [18] and to PageRank [2] in that each node distributes a constant amount of weight over its out-bound edges. The fewer pages a node points to, the more influence it can pass to its neighbors that it points to.

Our experiments were performed on a large subset of CiteSeer, that consisted of 132,210 documents and 461,170 citations. After the graph was normalized, we applied the hierarchical cut clustering algorithm of Figure 3. We started with the largest possible value of α that gave non-trivial clusters (not all singletons). Then we contracted those clusters, producing the graph of the next level. We clustered again with the largest possible value of α. These are the clusters of the second level and correspond to clusters of clusters of nodes. We repeated the process until we had only one cluster, containing the entire set of nodes.

By looking at the resulting clusters, we conclude that at lower levels the clusters contain nodes that correspond to documents with very specific content. For example, there are small clusters (usually with fewer than 10 nodes) that focus on topics like "LogP Model of Parallel Computation," "Wavelets and Digital Image Compression," "Low Power CMOS," "Nonholonomic Motion Planning," "Bayesian Interpolation," and thousands of others. Table 1 shows the titles of 3 documents of each of the clusters just mentioned. For each cluster, these three documents are, respectively, the heaviest, median and lightest linked within their clusters. We can see that the documents are closely related to each other. This was the case in all clusters of the lowest level.

LogP Model of Parallel Computation
LogP: Towards a Realistic Model of Parallel Computation
A Realistic Cost Model for the Communication Time in Parallel Programs
LogP Modelling of List Algorithms
Wavelets and Digital Image Compression
An Overview Of Wavelet Based Multiresolution Analyses
Wavelet Based Image Compression
Wavelets and Digital Image Compression Part I & II
Low Power CMOS
Low Power CMOS Digital Design
Power-Time Tradeoffs In Digital Filter Design And Implementation
Input Synchronization in Low Power CMOS Arithmetic Circuit Design
Nonholonomic Motion Planning
Nonholonomic Motion Planning: Steering Using Sinusoids
Nonholomobile Robot Manual
On Motion Planning of Nonholonomic Mobile Robots
Bayesian Interpolation
Bayesian Interpolation
Benchmarking Bayesian neural networks for time series forecasting
Bayesian linear regression

Table 1: CiteSeer data - example titles are from the highest, median, and lowest ranking papers within a community, thus demonstrating that the communities are topically focused.

For clusters of higher levels, we notice that they combine clusters of lower levels and singletons that have not been clustered with other nodes yet. The clusters of these levels (approximately after 10 iterations) focus on more general topics, like "Concurrent Neural Networks", "Software Engineering, Verification, Validation", "DNA and Bio Computing", "Encryption", etc.

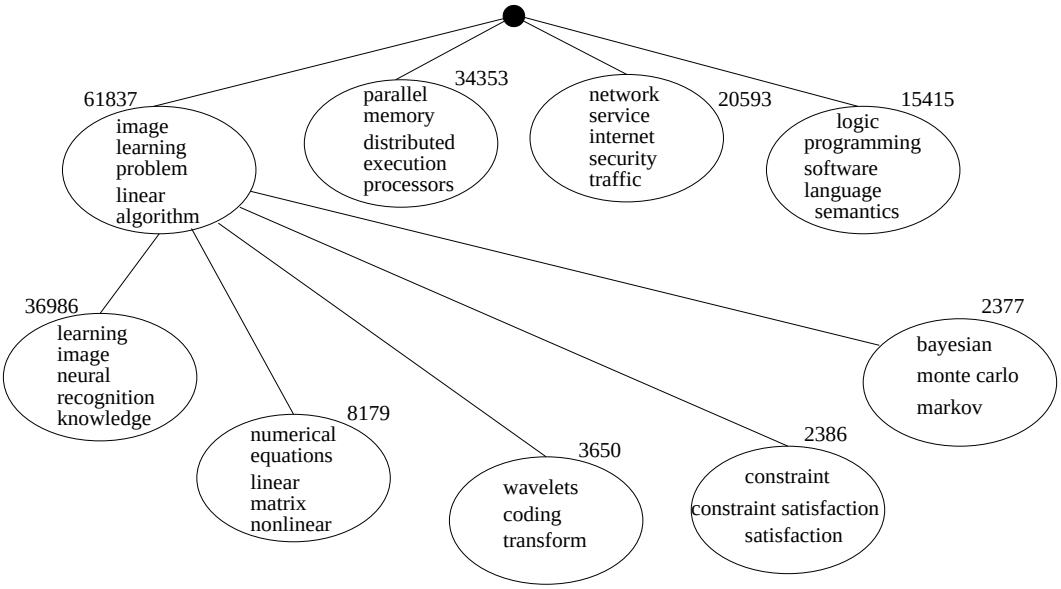

Figure 5: Top level clusters of CiteSeer. The sizes of each cluster are shown, as well as the top features for each cluster.

At the highest levels, the clusters become quite general, with broad topics covering entire areas, like "Networks", "Databases", and "Programming Languages". Figure 5 shows the highest non-trivial level, that consists of four clusters. In order to better validate our results, for each cluster, we have extracted text features (i.e. uni-grams and bi-grams) from the title and abstract of the papers. For each feature we calculate the expected entropy loss of the feature characterizing the pages within a cluster, relative to the pages outside that cluster. Figure 5 shows the five highest ranking features, for each cluster. Also, the sizes of the clusters are shown, as well as the second level for the largest cluster. Based on the features, we can see that the four major clusters in CiteSeer are on: "Algorithms and Graphics", "Systems", "Networks and Security", and "Software and Programming Languages".

As can be seen, the iterative cut clustering algorithm yields a high quality clustering, both on the large scale and on small scales, as supported by the split of topics on the top level, and the focused topicality on the lowest level clusters.

Also, because of the strict bounds that the algorithm imposes on the expansion for each cluster, it may happen that only a subgraph of G gets clustered into non-trivial clusters. This is also the case for the CiteSeer data, which is a very sparse graph (with an average node degree of only about 3.5). Thus, many of the initial documents do not get clustered at the lowest level. Nevertheless, when this is the case, the clusters that do get produced are still of very high quality and can serve as core communities or seed sets for other, less strict clustering algorithms.

5.2 Open Directory

The Open Directory Project [26], or *dmoz*, is a human edited directory of the Web. For our experiment we start off with the homepage of the Open Directory, `http://www.dmoz.org`, and crawl all web-pages up to two links away, in a breadth-first approach. This produces 1,624,772 Web pages. The goal is to see if and how much the clusters produced by our cut clustering algorithm coincide with those of the human editors.

In order to cluster these Web pages, we represent them as nodes of a graph, where the edges correspond to hyperlinks between them, but we exclude all links between web-pages of the same domain, because this

biases the graph. We also exclude any internal dmoz links. We will elaborate more on how internal links affect the clustering at the end of this section. The final graph is disconnected, with the largest connected component containing 638,458 nodes and 845,429 edges. The cut clustering algorithm can be applied to each connected component separately, or to the entire graph at once. For reasons of simplicity, and since the graph is not too large to handle, we choose the second approach. The average node degree of the graph is 1.47. Similar to the CiteSeer data, before applying the clustering algorithm, we make the graph undirected and normalize the edges over the out-bound links.

Applying the cut clustering algorithm for various values of α, we build the hierarchical tree of Figure 4 for this graph. Naturally, at the highest level we get the connected components of the initial graph. For the next two levels, only small pieces of the connected components are being cut off. At level four, the largest connected component gets broken up into hundreds of smaller clusters. There are many clusters that contain a few hundred nodes. The largest cluster contains about 68,073 nodes. Table 2 shows the largest 12 dmoz clusters and the top three features extracted for each of them. We notice that the top two clusters contain Web pages that refer to or are from the search engines Yahoo! and Altavista, reflecting their popularity and large presence in the dmoz category. The other ten clusters correspond to categories at the highest or second highest level of dmoz. In fact, 88 of the top 100 clusters correspond to categories of dmoz at the two highest levels. Regarding the coverage of the clusters, the top 250 clusters cover 65% of the total number of nodes, and the top 500 clusters cover over 85%.

Size	Feature 1	Feature 2	Feature 3
68,073	in yahoo	yahoo inc	copy yahoo inc
24,946	about altavista	altavista company	altavista reg
18,533	software search on	software	software top
17,609	economy search	economy regional	economy about
14,240	health about dmoz	health search on	health top
13,272	employment about dmoz	employment search on	employment top
11,867	tourism search	tourism search	tourism top
10,735	personal pages search	pages search on	personal pages
9,452	environment search	environment top	environment regional
8,289	entertainment search	entertainment about	entertainment top
7,813	culture top	culture about	culture search
7,421	sports search	sports regional	sport top

Table 2: Top 3 features for largest dmoz clusters.

Compared to the CiteSeer data, we conclude that the dmoz clusters are relatively smaller, but more concentrated. This is most likely the result of the extremely sparse dmoz graph. Still, the cut clustering algorithm is able to extract the most important categories of dmoz. A limitation of our algorithm is in finding broader categories, for example, corresponding only to the highest level of dmoz, but this is also a result of the sparseness of the graph. We have applied less restrictive clustering algorithms on top of the cut clustering algorithm, such as agglomerate and k-means algorithms, and were able to add intermediate level to the hierarchical tree that almost coincided with the top level categories of dmoz.

Regarding internal links, if we repeat the same experiment without excluding any internal links within dmoz or between two Web pages of the same site, the results are not as good. Internal links force clusters to be concentrated more around individual web sites. In particular, when a web site has many internal links, because of normalization, the external links get less weight and the results are skewed towards isolating the web site. We suggest that internal and external links should be treated independently. In our examples we ignored internal links entirely, but one could also assign some small weight to them by normalizing internal and external links separately, for example.

5.3 The 9/11 Community

In this example we did not cluster a set of Web pages, but were interested in identifying a single community, by using the cut clustering algorithm. The topic used to build the community was September 11, 2001.

Ten web pages were used as seed sites and the graph included web pages up to four links away that were crawled in 2002. The artificial sink was connected to the graph by edges with the smallest possible α value that broke it into more than one connected component. This resulted in a community of size 6257. Table 3 shows the top 50 features within that community. Again, even though the clustering algorithm considered only the link structure of the web graph, all top features, as extracted from the text, show the high topical concentration of the 9/11 community.

1.	A.terrorism	26.	E.attacks
2.	F.terrorist_attacks	27.	F.afghan
3.	F.bin_laden	28.	F.laden
4.	E.terrorism	29.	F.war_on_terrorism
5.	F.taliban	30.	A.terror
6.	F.on_terrorism	31.	F.afghanistan
7.	F.in_afghanistan	32.	F.homeland
8.	F.osama	33.	F.the_world_trade
9.	F.terrorism_and	34.	F.on_america
10.	F.osama_bin	35.	F.the_terrorist_attacks
11.	F.terrorism	36.	A.terrorism_http
12.	F.osama_bin_laden	37.	F.the_terrorist
13.	F.terrorist	38.	A.emergency
14.	E.afghanistan	39.	F.of_afghanistan
15.	F.against_terrorism	40.	F.the_attacks
16.	F.the_taliban	41.	F.the_september_11
17.	E.terrorist	42.	F.september_11th
18.	F.to_terrorism	43.	F.wtc
19.	A.state_gov	44.	F.of_terrorism
20.	F.anthrax	45.	A.attacks
21.	F.terrorist_attack	46.	A.www_state_gov
22.	F.world_trade_center	47.	F.sept_11
23.	F.the_attack	48.	T.terrorism
24.	F.terrorists	49.	F.attacks_on_the
25.	A.terrorist	50.	F.qaeda

Table 3: Top 50 features of the 9/11 community. A prefix A, E, or F indicates whether the feature occurred in the anchor text, extended anchor text (that is, anchor text including some of the surrounding text), or full text of the page, respectively.

Readers interested in more information on the 9/11 community should see:

`http://webselforganization.com/example.html`

which allows one to browse and search the community for specific words. The web site also shows some interesting 9/11 community members such as pages from the Howstuffworks web site [27] which describe (among other things) how nuclear bombs, airplane black boxes, and anthrax work.

6 Conclusions

We have shown that minimum cut trees, based on expanded graphs, provide a means for producing quality clusterings and for extracting heavily connected components. Our analysis shows how a single parameter, α, can be used as a strict bound on the expansion of the clustering while simultaneously serving to bound the inter-cluster weight as well. In this manner, we collapse a bicriterion into a single parameter framework.

While identifying communities is an NP-complete problem (see the appendix for more details), we partially resolve the intractability of the underlying problem by limiting what communities can be identified. More specifically, the cut clustering algorithm identifies communities that satisfy the strict bicriterion imposed by the choice of α, and no more. The algorithm is not even guaranteed to find all communities that obey the bounds, just those that happen to be obvious in the sense that they are visible within the cut tree

of the expanded graph. Hence, the main strength of the cut clustering algorithm is also its main weakness: it avoids computational intractability by focusing on the strict requirements that yield high quality clusterings.

All described variations of the cut clustering algorithm are relatively fast (especially with the use of the heuristic of Section 3.4), simple to implement, and give robust results. The flexibility of choosing α, and thus determining the quality of the clusters produced, is another advantage. Additionally, when α is not supplied, we can find all breakpoints of α fast [8].

On the other hand, our algorithms are limited in that they do not parameterize over the number and sizes of the clusters. The clusters are a natural result of the algorithm and cannot be set as desired (unless searched for by repeated calculations). Also note that cut clusters are not allowed to overlap, which is problematic for domains in which it seems more natural to assign a node to multiple categories (something we have also noticed with the CiteSeer data-set). This issue is a more general limitation of all clustering algorithms that produce hard or disjoint clusters.

Implementation-wise, maximum flow algorithms have significantly improved in speed over the past years ([10], [3]), but they are still computationally intense on large graphs, being polynomial in complexity. Randomized or approximation algorithms could yield similar results, but in less time, thus laying the basis for very fast cut clustering techniques of high quality.

References

[1] M. Bern and D. Eppstein. *Approximation Algorithms for Geometric Problems.* In D. S. Hochbaum, editor, *Approximation Algorithms for NP-Hard Problems.* PWS Publishing Company, 1996.

[2] S. Brin and L. Page. *Anatomy of a Large-Scale Hypertextual Web Search Engine*, Proc. 7th International World Wide Web Conference, 1998.

[3] C. S. Chekuri, A. V. Goldberg D. R. Karger, M. S. Levine, and C. Stein. Experimental Study of Minimum Cut Algorithms. In *Proc. 8th ACM-SIAM Symposium on Discrete Algorithms*, pages 324–333, 1997.

[4] B. V. Cherkassky and A. V. Goldberg. On Implementing Push-Relabel Method for the Maximum Flow Problem. *Algorithmica*, 19:390–410, 1997.

[5] B. Everitt. *Cluster analysis.* Halsted Press, New York, 1980.

[6] G. W. Flake, S. Lawrence and C. L. Giles. Efficient identification of web communities. In *Proceedings of the Sixth International Conference on Knowledge Discovery and Data Mining (ACM SIGKDD-2000)*, Boston, MA, 2000. ACM Press.

[7] L. R. Ford, Jr. and D. R. Fulkerson. *Flows in Networks.* Princeton Univ. Press, Princeton, NJ, 1962.

[8] G. Gallo, M. D. Grigoriadis and R. E. Tarjan. A Fast Parametric Maximum Flow Algorithm and Applications. SIAM Journal of Computing, Vol. 18, No. 1 (1989) 30-55.

[9] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman, New York, 1979.

[10] A. V. Goldberg. Recent developments in maximum flow algorithms. Technical Report No. 98-045, NEC Research Institute, Inc., Apr. 1998.

[11] A. V. Goldberg and R. E. Tarjan. A New Approach to the Maximum Flow Problem. *J. Assoc. Comput. Mach.*, 35:921–940, 1988.

[12] A. V. Goldberg and K. Tsioutsiouliklis. Cut tree algorthms: an experimental study. *J. Algorithms.* 38:51–83, 1 (2001).

[13] R. E. Gomory and T. C. Hu. Multi-terminal network flows. *J. SIAM*, 9:551–570, 1961.

[14] T. C. Hu. *Combinatorial Algorithms.* Addison-Wesley, Reading, MA, 1982.

[15] A. K. Jain and R. C. Dubes. *Algorithms and Clustering Data* Prentice-Hall, NJ, 1988.

[16] R. Kannan, S. Vempala, and A. Vetta. On Clusterings - Good, Bad and Spectral. In *IEEE Symposium on Foundations of Computer Science*, pages 367-377, 2000.

[17] R. M. Karp. *Reducibility among combinatorial problems.* In R. E. Miller and J. W. Thatcher (eds.), *Complexity of Computer Computations*, Plenum Press, New York, 85-103.

[18] Jon Kleinberg. Authoritative sources in a hyperlinked environment. *Proc. 9th ACM-SIAM Symposium on Discrete Algorithms*, 1998.

[19] S. Mancoridis, B. Mitchell, C. Rorres, Y. Chen, and E. Gansner. Using automatic clustering to produce high-level system organizations of source code. In Proceedings of the 6th Intl. Workshop on Program Comprehension, June 1998.

[20] H. Nagamochi, T. Ono, and T. Ibaraki. Implementing an Efficient Minimum Capacity Cut Algorithm. *Math. Prog.*, 67:297–324, 1994.

[21] Q. C. Nguyen and V. Venkateswaran. Implementations of Goldberg-Tarjan Maximum Flow Algorithm. In D. S. Johnson and C. C. McGeoch, editors, *Network Flows and Matching: First DIMACS Implementation Challenge*, pages 19–42. AMS, 1993.

[22] M. Padberg and G. Rinaldi. An Efficient Algorithm for the Minimum Capacity Cut Problem. *Math. Prog.*, 47:19–36, 1990.

[23] H. Saran and V. V. Vazirani. Finding k-cuts within twice the optimal. In *IEEE, editor, Proceedings of the 32nd Annual Symposium on Foundations of Computer Science*, pages 743–751, San Juan, Porto Rico, October 1991. IEEE Computer Society Press.

[24] Z. Wu and R. Leahy An optimal graph theoretic approach to data clustering: Theory and its application to image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, November 1993.

[25] `http://www.citeseer.com`

[26] `http://www.dmoz.org`

[27] `http://www.howstuffworks.com`

Appendix

We now provide a proof of Theorem 3.2 of Section 3. In fact, we shall prove a slightly more general theorem:

Theorem 6.1 *Define PARTITION INTO COMMUNITIES, or simply COMMUNITIES, as follows:*

> *PROBLEM:* ***PARTITION INTO COMMUNITIES***
> *INSTANCE: Undirected, weighted graph $G(V,E)$, real parameter $p \geq 1/2$, integer k.*
> *QUESTION: Can the vertices of G be partitioned into k disjoint sets $V_1, V_2, ..., V_k$, such that for $1 \leq i \leq k$, the subgraph of G induced by V_i is a community, that is, $\forall V_i, \forall u \in V_i$, $\sum_{v \in V_i} w(u,v) \geq p \cdot \sum_{v \in V} w(u,v)$?*

PARTITION INTO COMMUNITIES is NP-complete.

Proof. We will prove the above theorem by reducing BALANCED PARTITION, a restricted version of PARTITION, to PARTITION INTO COMMUNITIES.

Here are the definitions of PARTITION (from [9]) and BALANCED PARTITION.

> PROBLEM: **PARTITION**
> INSTANCE: A finite set A and a *size* $s(a) \in \mathbb{Z}^+$ for each $a \in A$.
> QUESTION: Is there a subset $A' \subseteq A$ such that $\sum_{a \in A'} s(a) = \sum_{a \in A - A'} s(a)$?

> PROBLEM: **BALANCED PARTITION**
> INSTANCE: A finite set A and a *size* $s(a) \in \mathbb{Z}^+$ for each $a \in A$.
> QUESTION: Is there a subset $A' \subseteq A$ such that $\sum_{a \in A'} s(a) = \sum_{a \in A - A'} s(a)$, and $|A'| = |A|/2$?

Both PARTITION and BALANCED PARTITION are well known NP-complete problems ([17] and [9]). To prove Theorem 6.1 we will reduce BALANCED PARTITION to COMMUNITIES.

First, it is easy to see that a given solution for COMMUNITIES is verifiable in polynomial time, by checking whether each node is connected to nodes of the same cluster by at least a fraction p of its total adjacent edge weight.

We transform the input to BALANCED PARTITION to that of COMMUNITIES. The input set C has cardinality $n = 2k$. We construct an undirected graph G with $2n + 4$ nodes as follows. First, n of G's nodes

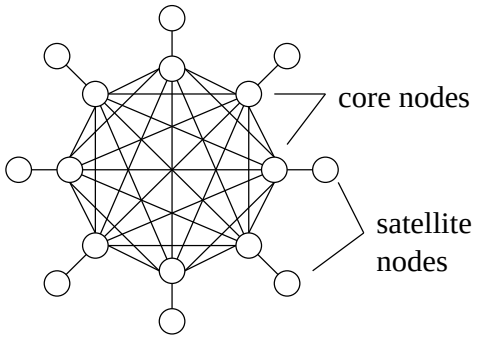

Figure 6: Core and satellite nodes forming the core graph.

form a complete graph K_n with all edge weights equal to w, where $\frac{s(a_{min})}{n} > w > 0$, and a_{min} has the smallest size $s(a_{min})$ among all elements a_i of A. Let us call these nodes the *core nodes* of the graph. Next, a *satellite node* is connected to each core node, with edge weight $w + \epsilon$, such that $\min\{\frac{w}{2}, \frac{|s(a_i)-s(a_j)|}{n}\} > \epsilon > 0$, for all a_i's and a_j's. The n satellite nodes have all degree 1 (Figure 6).

Now, we add two more nodes to the graph, which we call *base nodes*. Every core node is connected to both base nodes by edges of equal weight. We make sure that each core node is connected to the base nodes by a different $s(a_i)$ value. Finally, we add two more satellite nodes, one to each base node, with edges of weight ϵ. The final graph looks as in Figure 7.

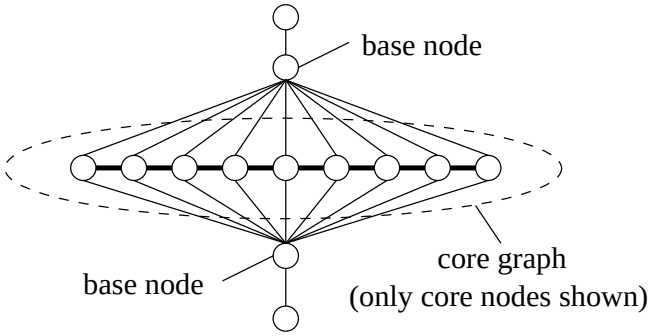

Figure 7: Final graph with base nodes.

Now, let us assume that COMMUNITIES is solvable in polynomial time. We transform an input of BALANCED PARTITION as mentioned above, and set $p = 1/2$ and $k = 2$.

Assume that COMMUNITIES gives a solution for this instance. Then, the following statements can be verified (we leave the proofs to the reader):

1. The solution must partition the core nodes into two non-empty sets, say S_1 and S_2.
2. The partitioning cannot place both base nodes in the same set.
3. S_1 and S_2 contain the same number of core nodes.
4. All satellite and core nodes satisfy the community property, that is, they are more strongly connected to other community members than to non-members.

Notice that the last statement does not include base nodes. Is it possible that their adjacent weight is less within their community? The answer depends on how the core nodes get divided. If the core nodes are divided in such a way that the sum of their adjacent a_i values is equal for both sets, then it is easy to verify that the base nodes do not violate the community property. In the opposite case, one of the two base nodes

will be more strongly connected to the nodes of the other community than to the nodes of its own community. Thus, the only possible solution for COMMUNITIES must partition all a_i's into two sets of equal sum and cardinality. But this implies that any instance to the BALANCED PARTITION problem can be solved by COMMUNITIES. And since BALANCED PARTITION is NP-complete and we can build the graph G and transform the one instance to the other in polynomial time, PARTITION INTO COMMUNITIES is NP-complete, as well. ■