Communities in Graphs — Project for CS 685

Ara Hayrapetyan David Kempe Martin Pál Zoya Svitkina

December 16, 2002

1 Introduction

Since its creation about ten years ago, the World Wide Web (WWW) has evolved not only into an extensive and sophisticated information storage, but also a large and complex social structure. In contrast to many of the more traditional information repositories, the WWW was created in an unstructured manner. Many users dispersed throughout the world authored a wide variety of web pages with different levels of sophistication, different topics and contents, often linking their pages to already existing ones. The result of this evolution process may therefore seem very unstructured.

Yet, the unplanned evolution of the WWW is far from being chaotic. Much can be inferred from the link structure of the web. The fact that the authors of a web page chose to create a link from their page to another one suggests that they believed that the target page contained information that was relevant to their interests. It is thus sensible to assume that the web pages about a given topic should be relatively interconnected, having many links amongst themselves and few links to unrelated pages. Therefore one might hope to be able to extract *communities*, collections of pages related to a given topic, using only the link structure of the WWW.

There are many reasons why it is important to identify communities in the World Wide Web. Some of the practical applications include focused search engines and automatic web portals. Communities are also an interesting target of sociological investigations interested in finding the ties between different social groups and detecting the emergence of new ones.

Companies such as Yahoo! [14] have compiled lists of topics that occur in the web. Web pages are classified into topics, which in turn are organized into a hierarchy: web pages pertaining to the same (or similar) topics can thus be regarded as a community. Classifying web pages is a tedious task when performed by humans, and it requires much human effort, time, and expertise. Ideally, one would like to be able to automatically extract the web pages related to a certain topic and perhaps even label them with text describing their content.

For extracting the WWW communities using solely the linkage information of the web, it is natural to view the web as a graph, where a node represents a web page, and a directed (or undirected) edge between two nodes represents the existence of a link from the first to the second. This framework obviously applies to other social networks as well, such as friendships or co-authorships.

In order to be able to approach this problem from an algorithmic perspective, the first goal that has to be achieved is to formulate a definition of a community in graph-theoretic terms, so that it approximates the common-sense judgement.

There have been several approaches to the definition of communities in graphs recently. Flake et al. [4, 5] define communities as sets such that each node in the set has at least a given fraction of its neighbors in the set. Kumar et al. [11] consider dense bipartite subgraphs as indications of communities, and expand such graphs into larger sets. Dense bipartite graphs can be considered as "hubs" and "authorities" on a given topic [10], and the Eigenvector-based techniques for finding such hubs and authorities have been used by Gibson et al. [6] to search for communities. Girvan and Newman [7] propose an algorithm that repeatedly deletes edges perceived to be "weak ties" in the sense of Granovetter [8], resulting in a hierarchical partition. Among other problems, this definition does not allow for communities to overlap. We will consider the approach by Flake et al. (with some modifications) more closely in the sequel.

2 Models of Communities and Overview of Results

In order to be able to find community structure in a graph, we have to define what exactly we mean by a "community". How useful a definition is will depend on two aspects: how much it corresponds to our intuitive notion of what a community is, and how tractable it is from a computational point of view.

Here, and throughout the rest of this report, we will always use the following notation. $G = (V, E)$ denotes an undirected graph. The degree of a node v is denoted by $d(v)$, and we extend the notation to write $d(S) = \sum_{v \in S} d(v)$ for sets S. For any two subsets S_1, S_2, we write $e(S_1, S_2)$ for the set of edges with one endpoint in S_1, and the other in S_2. Then, $e(S) = e(S, S)$ is the set of all edges with both endpoints in S. We extend the notation to nodes as well, and write $e(v, S)$ for the set of edges between v and the set S. The set of edges with exactly one endpoint in S, and the other in $\overline{S}$ is written as $\delta(S) = e(S, \overline{S})$. For much of the sequel, we assume that we have *capacities* $c(e)$ on the edges. The capacity of the edges between sets S_1, S_2 is denoted by $\gamma(S_1, S_2) = \sum_{e \in e(S_1, S_2)} c(e)$, and specifically for the cut $(S, \overline{S})$, we write $\gamma(S) = \gamma(S, \overline{S})$.

2.1 Dense Subgraphs

A natural first candidate for a definition of "community" are dense subgraphs. Given a graph G, and a non-empty node set $S \subseteq V$, we define the *density* of S to be $\frac{|e(S)|}{|S|}$. We can then consider sets with high density as communities. The intuition is that the higher density means that there is higher than average "mutual acquaintance" between nodes, indicative of social or topical structure.

The advantage of this definition is that it is tractable in the following sense: given a graph G, we can find, in polynomial time, a set S of maximum density. The algorithm is a relatively straightforward reduction to a minimum-cut problem [12]. In fact, the algorithm can be extended to solve the following problem. Given a subset X of nodes, find the densest subgraph S that contains all of X. We discuss the (small) modifications to the reduction in Section 3.

The envisaged applications for community-finding algorithms include very large graphs, such as the Web graph. For such graphs, even an algorithm consisting of only several maximum-flow computations might be prohibitively expensive. For that reason, we study the greedy approximation algorithm which always removes the lowest-degree node, and later returns the highest-density graph among all n graphs that were produced at intermediate steps. In Section 3, we show that this greedy algorithm is a 2-approximation, even in the case where a subset X of nodes has to be included in the dense set (and hence is never deleted by the greedy algorithm). We give a matching lower bound by showing that for any $\epsilon > 0$, there are graphs G on which the solution found by the greedy algorithm is worse than the optimum by a factor of $2 - \epsilon$.

The focus on dense subgraphs as communities has its drawbacks, however. The main one is that in graphs with very skewed degrees (such as the web graph: consider the difference between the degree of Yahoo! and a personal web site), high degree nodes will likely be included in all densest subgraphs. To see this, notice that if a (high-degree) node v is connected to at least half of the nodes in a set S, then $S \cup \{v\}$ is strictly denser than S. Intuitively, however, we would not consider v part of the community of S unless it, too, had most of its neighbors in S.

A possible approach to deal with this problem would be to seek *small* dense subgraphs, e.g. subgraphs whose number of nodes is bounded by some constant k. Unluckily, this change (which is far from certain to work) makes the problem intractable, since we are then studying the densest k-subgraph problem [1, 2, 3]. This problem is known to be NP-complete (via a simple reduction from the CLIQUE problem), and while no approximation hardness results are known, the best known approximation ratio is $O(n^{1/3-\epsilon})$ for some $\epsilon > 0$. If we are going to make the problem hard, we might as well use a more intuitive model in the process.

2.2 Inclusion-Exclusion Models

We mentioned above that one drawback to the "dense subgraph" definition of communities is the inclusion of high-degree nodes. We would wish to force nodes not only to contribute many edges inside the set, but also to have few edges leaving the set. This idea is captured in a proposed definition by Flake et al. [4, 5]. The definition calls a set S an α-community if each node $v \in S$ satisfies $|e(v, S)| \geq \alpha \cdot d(v)$, i.e. if each member of S has at least an α fraction of its neighbors in S. The goal is then to find non-empty sets S which are α-communities for large values of α.

In this sense, the entire graph G is always an α-community for all $\alpha \leq 1$, and as no non-empty set can be an α-community for $\alpha > 1$, the entire graph will always be an optimal solution. We are therefore really interested in finding *small* α-communities. Unluckily, even approximating the smallest α-community within a factor of $O(\log n)$ is NP-hard, making this definition less useful in practice. This result was proved in [9] — for completeness, we give the proof in Appendix A. More generally, the definition suffers from extremely high sensitivity to small changes in the graph: adding or removing a single edge can change the largest α attained by a given set S dramatically. Similarly, the size of the smallest α-community may change a lot. This suggests that even bicriteria approximations might be hard to obtain (and that the lower bound of $\Omega(\log n)$ for the approximation hardness might not be tight).

We therefore propose the following notion of an *average α-community*: a set S is an average α-community if $\frac{|e(S)|}{d(S)} \geq \alpha$. Notice that if S is an α-community, it is also an average $\frac{\alpha}{2}$-community. The advantage of the new definition is that it is less sensitive to small changes: adding or removing a single edge cannot change the value $\frac{|e(S)|}{d(S)}$ for a large set S by very much.

This definition looks very similar to the one for dense subgraphs, but penalizes for the inclusion of high-degree nodes. There is of course no intrinsic reason (besides intuitive appeal), why the penalty incurred by a node v should be exactly the degree $d(v)$. We can thus generalize both the densest subgraph and the average community problem by assigning weights $w(v)$ to nodes, and finding a set S maximizing $\frac{|e(S)|}{w(S)}$. With a simple reduction, we will show in Section 3 that the versions with general weights and average communities are equivalent so long as the weights $w(v)$ are polynomial in the size of the graph.

When the weights are sufficiently large (for instance, when $w(v) = d(v)$), then the best α will still be obtained by the entire graph, as with the definition by Flake et al. Hence, we are again trying to find *small* sets S with high ratios. The equivalence between the problem with general weights and the one with weights equal to degrees, together with the NP-completeness of the densest k-subgraph problem, then implies that finding the smallest average α-community is still an NP-complete problem.

So why do we consider this new definition preferable? We believe that it can likely be approximated much better than the definition given be Flake et al. Certainly, the reduction showing approximation hardness in the Flake et al. model cannot be easily adapted to average α-communities. Furthermore, as we see in Section 3, the standard min-cut reduction for the densest subgraph problem can be adapted to find average α-communities, even with general weights. In that sense, any graph cut of capacity bounded by some constant (that is a result of the reduction) is an average α-community. Hence, we have reduced the problem further, to studying small cuts in graphs.

2.3 Minimum Size Bounded Capacity Cuts

The *minimum size bounded capacity cut* problem (MSBCC) is defined as follows: given a graph G with edge capacities $c(e)$, specified source and sink vertices s and t, and a capacity constraint C, find a cut $(S, \overline{S})$ of capacity $\gamma(S) = \sum_{e \in \delta(S)} c(e)$ at most C, such that $s \in S, t \in \overline{S}$, and the size of S is minimized. By the reduction given in Section 3, this problem is NP-complete.

We believe that the MSBCC problem is worth studying for several reasons: First, it is a very natural graph-theoretic problem, interesting in its own right, and may provide insights into the structures of cuts in graphs.

Second, any approximation guarantees for this problem translate directly into approximation guarantees for the general version of the average α-community problem. In particular, when the weights are the degrees, it gives us small communities in an intuitive sense resembling the definition of Flake et al. And when all weights are $w(v) = 1$, then we obtain different approximation guarantees for the densest k-subgraph problem. Specifically, this would allow us to find an approximately smallest subgraph of at least a given density. While this direction for approximation does not seem to have received as much attention in the literature as approximating the density given a fixed size constraint, it is nevertheless a natural problem, for which results may prove interesting. Our main result in this report is the following Theorem:

Theorem 2.1 *1. For graphs G of bounded treewidth, the smallest s-t cut with a given capacity constraint can be computed in strongly polynomial time.*

2. *For arbitrary graphs G, we obtain the following bicriteria approximation: We can find, in polynomial time, an s-t cut S of capacity at most $2C$, such that S contains at most twice as many nodes as the smallest s-t cut of capacity at most C.*

The proof of Theorem 2.1 is given in Sections 4 and 5. The algorithm for bounded treewidth graphs uses dynamic programming, and the bicriteria approximation considers the LP-relaxation of an integer programming formulation of the problem. We prove that it is not necessary to solve the linear program, as its optimum solution can be found with one parametric max-flow computation.

Unluckily, the bicriteria approximation does not give a similar guarantee for the average α-community problem (or the densest subgraph problem), because the reduction in Section 3 introduces a large additive constant into the objective function. We believe that these partial results may be of interest nevertheless, and hope to continue working on obtaining single-parameter approximation results.

3 Smaller results

In this section, we prove some results about finding densest subgraphs and average communities using greedy and graph-cut based approaches, and prove the NP-completeness of the minimum size bounded capacity cut problem.

3.1 Finding dense subgraphs greedily

Given a graph $G = (V, E)$, a constant α, and a (possibly empty) subset of *required nodes* $X \subseteq V$, we wish to find a set $S \subseteq V$ of edge density $\frac{|e(S)|}{|S|} \geq \alpha$, such that $X \subseteq S$. The exact algorithm for this problem, which is based on graph cuts, will be discussed shortly. But first, we give an approximate greedy algorithm with faster running time, which is important if we want to consider graphs as large as the WWW.

The greedy algorithm for the densest subgraph without the required nodes was shown to be a 2-approximation previously [9]. Here, we generalize the algorithm to include the required nodes and prove that it still returns a result with density within a factor of 2 of the optimum.

Starting with a graph $G = (V, E)$ and a set $X \subseteq V$ of required vertices, the algorithm iteratively removes the node $v \notin X$ with the lowest degree:

1. `Start with` $G_n = G$, `and` $k = n$.
2. `While` $G_k \supsetneq X$, `let` $G_{k-1} \leftarrow G_k \setminus \{v\}$ `and` $k \leftarrow k - 1$,
   `where` v `is a node in` $G_k \setminus X$ `with minimal degree` $d_{G_k}(v)$,
   `and go to iteration` $k - 1$.
3. `Return the graph` G_k `maximizing` $\frac{|e(G_k)|}{|G_k|}$.

Theorem 3.1 *Let $\delta = \max_{S \supseteq X} \frac{|e(S)|}{|S|}$, and r the edge density for the graph returned by the greedy algorithm. Then, $r/\delta \geq \frac{1}{2}$.*

Proof. Let S be an optimal solution, and let k be such that G_k is the graph right before a vertex from S is removed by the greedy algorithm for the first time. Call this vertex v. By the optimality of S, we have that

$$\frac{|e(S \setminus \{v\})|}{|S|-1} = \frac{|e(S)| - d_S(v)}{|S|-1} \leq \frac{|e(S)|}{|S|}.$$

By rearranging, we obtain that $d_S(v) \geq \frac{|e(S)|}{|S|}$. Since v is the lowest-degree node in $G_k \setminus X$, and $X \subseteq S$, we have $d_S(u) \geq \frac{|e(S)|}{|S|}$ for all $u \in G_k \setminus S$. Because $G_k \supseteq S$, we know that $d_{G_k}(u) \geq d_S(u)$ for all $u \in S$. The edge density of G_k is then

$$\frac{|e(G_k)|}{|G_k|} = \frac{\sum_{u \in G_k} d_{G_k}(u)}{2|G_k|} = \frac{\sum_{u \in S} d_{G_k}(u) + \sum_{u \in G_k \setminus S} d_{G_k}(u)}{2|G_k|}$$

Using the above inequalities,

$$\frac{|e(G_k)|}{|G_k|} \geq \frac{\sum_{u \in S} d_S(u) + \sum_{u \in G_k \setminus S} \frac{|e(S)|}{|S|}}{2|G_k|} = \frac{2|e(S)| + |G_k \setminus S| \frac{|e(S)|}{|S|}}{2|G_k|} \geq \frac{|e(S)|}{|S|} \cdot \frac{|S| + |G_k \setminus S|}{2|G_k|} = \frac{|e(S)|}{2|S|}$$

The density of the graph returned by the greedy algorithm is no smaller, as G_k was a candidate, which completes the proof of the approximation ratio. ■

Now, we give a lower bound for this greedy algorithm by constructing a class of graphs on which the ratio of greedy to optimal solution is $r/\delta \leq \frac{1}{2} + \epsilon$ for any $\epsilon > 0$. Such a graph consists of a clique A of size n, α nodes, each of which is connected to all nodes of A, but which are not connected to each other (we call these the *satellite nodes*), and β cliques of size $n+2$, which are not connected to anything else.

The subset with the highest edge to node ratio in this graph is A together with all satellite nodes connected to it. The ratio is

$$\frac{\alpha n + \binom{n}{2}}{\alpha + n} = \frac{2\alpha n + n^2 - n}{2(\alpha + n)} = \frac{n}{2} + \frac{n(\alpha - 1)}{2(\alpha + n)},$$

which tends to n as α goes to ∞.

Since the satellite nodes have the lowest degrees, the greedy algorithm will remove all of them first, then proceed to nodes in A, and then remove the nodes in the $(n+2)$-sized cliques. So the choices for the output graph that it has are:

The entire graph G:

$$\frac{e(G)}{|G|} = \frac{\beta\binom{n+2}{2} + \alpha n + \binom{n}{2}}{\beta(n+2) + \alpha + n} = \frac{\beta(n+2)(n+1) + 2\alpha n + n(n-1)}{2\beta(n+2) + 2(\alpha + n)}$$

which goes to $\frac{n+1}{2}$ if $\beta = \alpha^2$ and $\alpha \to \infty$, and r/δ can be arbitrarily close to $\frac{1}{2}$ for a suitable choice of n.

The graph with $\alpha - \alpha'$ satellite nodes removed: We obtain the same equation as above, with α replaced by α'. The limit is the same because the dominating terms do not change.

k of the cliques of size $n+2$: The ratio is $\frac{k\binom{n+2}{2}}{k(n+2)} = \frac{n+1}{2}$, like in the above cases.

Since it can be shown that a smaller clique gives a worse ratio than a larger clique, and including two disconnected components gives a ratio that is no better than the best of the two ratios, we conclude that there exist numbers α, β, and n for which the above construction has $r/\delta \leq \frac{1}{2} + \epsilon$.

3.2 Finding dense subgraphs using graph cuts

Here, we give a version of the min-cut algorithm [12] for finding densest subgraphs, modified to take into account the required nodes.

The original algorithm proceeds by computing a minimum s-t cut on a modified graph, which is obtained by adding two nodes to the graph G, an artificial source s and an artificial sink t, and connecting each node v of the original graph to s with an edge of capacity $d(v)$, and to t with an edge of capacity 2α. Any s-t cut (S,T) in this graph has capacity $C = d(T) + 2\alpha|S| + e(S,T)$. Observing that $|E| = |e(S,S)| + |e(S,T)| + |e(T,T)|$ and $d(T) = |e(S,T)| + 2|e(T,T)|$, the capacity of the cut can be rewritten as $C = 2|E| - 2|e(S,S)| + 2\alpha|S|$. Therefore, if there exists a cut with capacity $C \leq 2|E|$, then $2\alpha|S| - 2e(S,S) \leq 0$, i.e. $\frac{|e(S)|}{|S|} \geq \alpha$, and therefore S is the required dense set. Since the min-cut algorithm finds the cut with the minimum capacity, if there exists at least one cut satisfying the above inequality, one such cut will be found. Conversely, if a subset with the required ratio exists, the corresponding cut will have capacity satisfying the inequality $C \leq 2|E|$.

If the set X is non-empty, the construction of the modified graph is identical, except that the capacity of edges between $v \in X$ and s is set to infinity (or a sufficiently high number), to ensure that $X \subseteq S$ in the resulting cut. Since the capacity of these edges does not appear in the equations above, an identical argument shows that if a small-capacity cut is found, then the identified subset of nodes satisfies the condition $\frac{|e(S)|}{|S|} \geq \alpha$, and, conversely, if such a subset (containing X) exists, one will be found.

3.3 Average α-community and generalized weights model

The densest subgraph and the average community models introduced in Section 2 can be generalized to the problem of finding a set S maximizing $\frac{|e(S)|}{w(S)}$ in a graph where nodes have weights $w(v)$. Then the densest subgraph problem is a special case with $w(v) = 1$, and the average community problem is a special case with $w(v) = d(v)$.

When the weights of nodes are equal to their degrees, the graph as a whole achieves the ratio of $\frac{|e(S)|}{d(S)} = \frac{1}{2}$, which is the maximum possible for any node set. Therefore, what we really want to find are *small* subsets maximizing $\frac{|e(S)|}{d(S)}$. Here we show that finding a subset of size no greater than k with weights equal to degrees is equivalent to finding it with general weights.

Clearly, the average community problem can be reduced to the generalized weights problem by setting the weights equal to the degrees. We provide a reduction in the other direction: given a graph G, we construct a modified graph G', such that maximizing $\frac{|e(S)|}{d(S)}$ in G' maximizes $\frac{|e(S)|}{w(S)}$ in G, given that the size of the sets considered is limited by k. The reduction works if the weights in G are integers, and if no non-isolated node v has $w(v) = 0$. This is a polynomial-time reduction if the weights are polynomial in the size of G.

The general idea of the reduction is to increase the degrees of nodes in G to match their weights. First, the weights on all nodes are scaled up by a multiplicative factor, such that no node's degree exceeds its weight. This does not change the optimal solution, since the objective function is scaled by the same factor for all candidates.

A big clique is added to the graph, and any node v whose weight exceeds its degree by h is connected to h different nodes in the clique. Now, for any subset of the original nodes, the objective function that uses scaled weights is equal to the objective function that uses degrees. The only thing left to show is that no node from the clique will be included in the optimal solution. We achieve this by making the clique sufficiently large, so that any edge of the original graph with its two endpoints is a better solution than any subset which includes a node from the clique. Let m denote the maximum node weight in the graph. We make the size of the clique $K > 2m \cdot k^2$, so each node in the clique has degree at least K. Then, $\frac{k^2}{K} < \frac{1}{2m}$, so $\frac{1}{2m}$ is an upper bound on the ratio achieved by any set T including a node from the clique. But this is smaller than any single edge incident to two nodes not in the clique, since that has ratio at least $\frac{1}{2m}$.

The minimum cut construction described above for the densest subgraph problem can be adapted for the general-weight average α-community problem. The only modification is that the edge between node v and the sink receives the capacity of $2\alpha w(v)$, instead of just 2α. Then, the capacity of any s-t cut is $C = 2|E| - 2e(S, S) + 2\alpha w(S)$, and a cut of capacity $C \leq 2|E|$ determines a set S such that $\frac{|e(S)|}{w(S)} \geq \alpha$.

Imposing a rigid size limit on communities seems artificial. Instead, we want to find the smallest communities satisfying a given ratio α. In our construction, finding the smallest average α-community corresponds to finding an s-t cut of capacity at most $2|E|$, such that the number of nodes on the source side of the cut is minimized. Motivated by this, we study this problem, which we call the Minimum Size Bounded Capacity Cut, on general graphs. But first, we show that it is NP-complete.

3.4 Minimum Size Bounded Capacity Cut is NP-complete

The minimum size bounded capacity cut problem is defined in Section 2. Here, we restate it as a decision problem and give an NP-completeness proof.

Minimum Size Bounded Capacity Cut: Given a graph G with edge capacities $c(e)$, specified source and sink vertices s and t, a capacity constraint C and a constant k, decide if there is a cut $(S, \overline{S})$ of capacity at most C, such that $s \in S, t \in \overline{S}$, and $|S| \leq k$.

Membership in NP is obvious. The NP-completeness reduction is from the k-CLIQUE problem. Given a graph $G = (V, E)$ and a number k, we construct the following instance of MSBCC, such that the positive answer to it implies the existence of a k-clique in G, and vice versa.

A graph G' is constructed from G in the same way as for finding densest subgraphs (as described in Section 3.2), with $\alpha = \frac{k-1}{2}$. The instance of MSBCC then consists of the graph G', the capacity constraint $2|E|$, and the size limit $k + 1$.

As shown earlier for this construction, the existence of the required cut on G' implies the existence of a set $S \subseteq V$ of size at most k, such that $\frac{|e(S)|}{|S|} \geq \alpha = \frac{k-1}{2}$. Therefore, $|e(S)| \geq \frac{k-1}{2}|S|$. Since no node set of size less than k can have this many edges, we know that $|S| = k$, and therefore, S is a clique. Conversely, if G contains a clique of size k, the cut in G' with the source side which includes only the source node and the clique will satisfy the conditions of the MSBCC instance.

4 Bounded Treewidth

In this section, we give a dynamic programming algorithm for solving the MSBCC problem in polynomial time for graphs of bounded treewidth, thus proving the first part of Theorem 2.1. We first give an algorithm for trees, to give the intuition, and then extend it to graphs of bounded treewidth.

4.1 An algorithm for trees

Throughout, we will think of the tree as rooted at s, and all edges directed away from s. Then, the terms "the tree rooted at v", "the edge into v" etc. have well-defined meanings.

If all the edge capacities in the tree are 1, then all we need to do is sort the subtrees $T_1, T_2, \ldots, T_d$ rooted at s by increasing size, and include all of $T_1, \ldots, T_{d-C}$ in the set S. The resulting cut has capacity $d - (d - C) = C$, and S is the smallest such set.

Now, let us consider the case of general edge capacities. Consider the tree T_v rooted at a node v, together with the edge e_v into v. We define the quantity a_v^k to be the smallest total edge capacity that must be cut when at most k nodes of T_v can be included. Notice that $a_v^0 = c(e_v)$. We can use dynamic programming to construct this table in a bottom-up fashion.

- If we include a leaf v, we need not cut any edges; otherwise, we must cut the edge e_v into the leaf. Notice that if the leaf is the sink t, then we do not have the option of including it, and must cut the edge into it. Hence, for leaves v, we have that

$$\begin{aligned} a_v^0 &= c(e_v) \\ a_t^k &= c(e_v) \quad \text{for all } k \\ a_v^k &= 0 \quad \text{for all } k > 0. \end{aligned}$$

- For an internal node v with children $v_1, \ldots, v_d$, we can either cut the edge e_v into v, or otherwise include v and solve the problem recursively for the children of v. Again, we may not include v if it is equal to the sink t. Hence, we have the recurrence

$$\begin{aligned} a_v^0 &= c(e_v) \\ a_t^k &= c(e_v) \quad \text{for all } k \\ a_v^k &= \min\left(c(e_v), \min_{k_1 \geq 0, \ldots, k_d \geq 0 : \sum k_i = k-1} \sum_i a_{v_i}^{k_i} \right) \quad \text{for } k > 0. \end{aligned}$$

  To avoid computing the minimum over the exponentially many possible choices of values k_i, we can again use dynamic programming, this time over the degree d. Specifically, we can define

$$b_v(k, j) \quad = \quad \min_{k_1 \geq 0, \ldots, k_j \geq 0 : \sum k_i = k} \sum_{i=1}^{j} a_{v_i}^{k_i}$$

  for all $1 \leq j \leq d$. Then, we can compute the $b_v(k, j)$ via dynamic programming as $b_v(k, 1) = a_{v_1}^k$, and $b_v(k, j+1) = \min_{0 \leq k' \leq k}(b_v(k', j) + a_{v_{j+1}}^{k-k'})$. Finally, we have that $a_v^k = \min(c(e_v), b_v(k-1, d))$.

Once we have computed a_s^k at the source s for all values of k, we simply pick the smallest $\hat{k}$ such that $a_s^{\hat{k}} \leq C$. A simple inductive proof shows that the algorithm actually maintains the initial definition of a_v^k as an invariant. In particular, this implies that $\hat{k}$ is the size of the minimum size bounded capacity cut. We can also use the table values to actually reconstruct the corresponding set S. At each node v (of degree d), the computation required to find a_v^k is $O(kd) = O(n^2)$, and hence, the overall running time is $O(n^4)$.

4.2 Graphs with bounded treewidth

Recall that a graph $G = (V, E)$ has treewidth θ if there exists a tree T, and subsets $V_w \subseteq V$ of nodes associated with each vertex w of T satisfying the following properties:

1. For every node $v \in V$ there exists a V_w containing it.
2. For every edge $e = (u, v) \in E$ there exists a set V_w containing both u and v.
3. If $\hat{w}$ lies on the path between w and w' in T, then $V_w \cap V_{w'} \subseteq V_{\hat{w}}$.
4. $|V_w| \leq \theta + 1$ for all vertices w of the tree T.

The pair $(T, \{V_w\})$ is called a *tree decomposition* of G, and the sets V_w will be called *pieces*. It can be shown that for any two neighboring vertices w and w' of the tree T, the deletion of $V_w \cap V_{w'}$ from G disconnects G into two components, just as the deletion of the edge (w, w') would disconnect T into two components.

For our algorithm, we assume that we are given an (optimal) tree decomposition $(T, \{V_w\})$ of our graph G. Furthermore, we associate with each node $w \in T$ a set $E_w \subseteq E \cap (V_w \times V_w)$ of edges both of whose endpoints lie in V_w, such that each edge appears in exactly one set E_w (this is a simple way to ensure that we do not count the cost of an edge multiple times). We will identify some node r of the tree T with $s \in V_r$ as being the *root*, and consider the edges of T as directed away from r.

We would like to mimick the algorithm for trees, by expressing quantities corresponding to a_v^k via recursive relations. This time, however, we need to allow for more parameters: the decision to include (or not include) a node $v \in V_w$ at some node w of the tree decomposition may affect the cost at nodes below it in the tree. In particular, we must be consistent about whether or not a node v is included. We will therefore think in terms of a top-down traversal of the tree. When a node v is encountered for the first time in a piece, i.e. $v \in V_w$, we have to make a decision whether or not v is included, and this decision must be respected by all descendants of w. A priori, this would force us to keep all subsets U of nodes as parameters in the table $a_w^k(U, U')$ — here, property 3 of the tree decomposition is crucial, for it allows us to only maintain the intersection of $U \cap V_w$. This is because no node that is not in V_w can be in the pieces both for an ancestor and a descendant of w.

We now formally define the quantity $a_w^k(U, U')$ for tree nodes $w \in T$. We assume that $U, U' \subseteq V_w$ are arbitrary sets of nodes included in the piece at w. Let $W \subseteq T$ be the set of nodes in the subtree rooted at w, $E_W = \bigcup_{w \in W} E_w$, and $V_W = \bigcup_{w \in W} V_w$. We define $a_w^k(U, U')$ to be the minimum capacity of edges from E_W that must be cut by any set $S \subseteq V_W$ such that $S \supseteq U$, $S \cap U' = \emptyset$, the sink t is not included in S (i.e. $t \notin S$), and $|S \setminus U| \leq k$. The minimum size of any cut of capacity at most C is then the smallest $\hat{k}$ for which $a_r^{\hat{k}-1}(\{s\}, \emptyset) \leq C$.

Our goal is to compute the values $a_w^k(U, U')$ in a bottom-up fashion using dynamic programming. At any stage, we will be taking a minimum over all subsets that meet the constraints imposed by the size k and the sets U, U'. We therefore write $\mathcal{S}_w^k(U, U') = \{S \mid U \subseteq S \subseteq V_w, S \cap U' = \emptyset, t \notin S, |S| \leq k\}$. In addition to forcing the inclusion of U, and disallowing U', we also ensure that we can never include the sink node t. Notice that the size of $\mathcal{S}_w^k(U, U')$ is $O(2^\theta)$. The cost incurred by cutting edges assigned to w is denoted by $\beta_w(U) = \sum_{e \in E_w \cap e(U, V_w \setminus U)} c(e)$. (Notice that $\beta_w(U)$ can be computed quickly.) The dynamic programming recurrence can now be stated as follows:

- If w is a leaf node, then we can include up to k additional nodes, so long as we do not violate the constraints imposed by the sets U and U'. Hence,

$$a_w^k(U, U') = \min_{S \in \mathcal{S}_w^k(U, U')} \beta_w(S).$$

- For an internal node w, let $w_1, \ldots, w_d$ denote its children. We can include an arbitrary subset of nodes, so long as we add no more than k nodes and do not violate the constraints imposed by the sets U and U'. The remaining additional nodes can then be divided between the children of w in any way desired. Once we have decided to include (or to not include) a node $v \in V_w$, we are forced to stick with that decision for all children, i.e. we obtain different sets as constraints for the children. Notice that any node v included in the pieces at two descendants of w must also be in the piece at w itself by property 3 of a tree decomposition. Also, by the same property, any node v from V_w that is not in V_{w_i} (for some child w_i of w) will not be in the piece at any descendant of w_i, and hence the information about v being forbidden or forced to be included is irrelevant in the subtree rooted at w_i. We hence have the following recurrence:

$$a_w^k(U, U') = \min_{S \in \mathcal{S}_w^k(U,U')} \min_{(k_i): \sum k_i = k - |S \setminus U|} \left(\beta_w(S) + \sum_i a_{w_i}^{k_i}(S \cap V_{w_i}, (V_w \setminus S) \cap V_{w_i}) \right).$$

Once we have fixed a set S, we can avoid considering the (exponentially many) combinations of k_i values in exactly the same way as for the tree case, by dynamic programming over the number j of children considered.

By induction over the tree, we can prove that this recursive definition actually coincides with the initial definition of $a_w^k(U, U')$, and hence that the algorithm is correct. The computation of a table entry $a_w^k(U, U')$ takes time $O(d \cdot k \cdot 2^\theta) = O(n^2 \cdot 2^\theta)$. For each node, we need to compute $O(n \cdot 4^\theta)$ entries, so the total running time is $O(n^4 \cdot 8^\theta)$, and the space requirement is $O(n^2 \cdot 4^\theta)$.

5 A Bicriteria Approximation

In this section, we present and analyze a bicriteria approximation algorithm for the MSBCC problem. Given a capacity C, our algorithm produces a cut $(S, V \setminus S)$ of capacity at most $2C$, such that the size of S is at most twice the size of the optimum cut with capacity bounded by C.

We first give a simple LP-based randomized rounding algorithm and then show how to derandomize it. Finally, by using Lagrangian relaxation and parametric flow techniques, we show how to avoid solving the LP.

5.1 The algorithm

We can formulate the problem as an integer program as follows. We will have a variable x_v for every vertex $v \in V$, and a variable y_e for every directed edge e.

$$\begin{array}{lllll} \text{Minimize} & \sum_{v \in V} x_v & & & \\ \text{subject to} & x_s & = & 1 & \\ & x_t & = & 0 & \\ & y_e & \geq & x_u - x_v & \text{for all } e = (u,v) \in E \\ & \sum_{e \in E} y_e \cdot c(e) & \leq & C & \\ & x_v, y_e & \in & \{0,1\} & \end{array} \tag{1}$$

We obtain a linear programming relaxation by replacing the constraints $x_v, y_e \in \{0,1\}$ by $x_v, y_e \geq 0$. Our algorithm can then be stated as follows:

1. Let $(\hat{x}, \hat{y})$ be an optimal solution to the LP (1).
2. Choose $\lambda \in [\frac{1}{2}, 1]$ uniformly at random.
3. Let $S = \{v \mid \hat{x}_v \geq \lambda\}$, and output S.

Theorem 5.1 *1. The above algorithm outputs a set S of size at most twice the LP objective value.*

2. The expected capacity of the cut $\delta(S)$ is at most $2C$.

Proof. 1. For each $v \in S$ we have that $\hat{x}_v \geq \lambda \geq \frac{1}{2}$. Hence, $\sum_{v \in V} \hat{x}_v \geq \sum_{v \in S} \hat{x}_v \geq \frac{1}{2}|S|$.

2. Consider an experiment in which we choose $\lambda \in [0, 1]$ uniformly at random and then define the set S as in step 3 of the algorithm. In this experiment, let X_e be an indicator random variable that is 1 if the edge $e = (u, v)$ is cut (i.e. $u \in S$, $v \notin S$), and 0 otherwise. Note that

$$\text{Prob}[X_e = 1] = \max(\hat{x}_u - \hat{x}_v, 0) \leq \hat{y}_e.$$

Hence, by linearity of expectation,

$$\mathrm{E}\left[\gamma(S)\right] \;=\; \textstyle\sum_{e=(u,v)\in E} c(e)\cdot \mathrm{Prob}[X_e=1] \;\le\; \sum_{e\in E} c(e)\hat{y}_e \;\le\; C.$$

By considering conditional expectations, we can write the expectation as

$$\begin{aligned}\mathrm{E}\left[\gamma(S)\right] &= \mathrm{E}\left[\gamma(S)\mid\lambda<\frac{1}{2}\right]\cdot\mathrm{Prob}[\lambda<\frac{1}{2}]+\mathrm{E}\left[\gamma(S)\mid\lambda\ge\frac{1}{2}\right]\cdot\mathrm{Prob}[\lambda\ge\frac{1}{2}]\\ &\ge \frac{1}{2}\mathrm{E}\left[\gamma(S)\mid\lambda\ge\frac{1}{2}\right],\end{aligned}$$

as $\gamma(S)$ is nonnegative and hence $\mathrm{E}\left[\gamma(S)\mid\lambda<\frac{1}{2}\right]\ge 0$. Hence, we proved that $\mathrm{E}\left[\gamma(S)\mid\lambda\ge\frac{1}{2}\right]\le 2C$. Note that this expectation is exactly the expected capacity of the cut produced by our algorithm. ∎

Instead of choosing λ randomly, we can try all possibilities and output the smallest cut we encounter. Since there are at most $|V|$ different values of $\hat{x}_v$, we need to try at most $|V|$ different cuts.

5.2 Lagrangian Relaxation

Here, we show how to find an optimum solution to the LP (1) using a single parametric flow computation. We use a Lagrangian Relaxation technique. We remove the constraint $\sum y_e \le C$ and put it into the objective function instead (α is a constant).

$$\begin{array}{lllll}\text{Minimize} & \alpha\cdot\sum_{v\in V} x_v+\sum_{e\in E} y_e\cdot c(e) & & & \\ \text{subject to} & x_s & = & 1 & \\ & x_t & = & 0 & \\ & y_e & \ge & x_u-x_v & \text{for all } e=(u,v)\in E\\ & x_v,y_e & \in & \{0,1\} & \end{array} \tag{2}$$

We claim that for any fixed value of α we can solve the IP (2) by solving a minimum cut problem. First, we claim that the LP-relaxation of this IP has an integer optimum solution. Consider an optimum solution (x^*,y^*) and suppose that it is not integral. Construct a random integral solution $(\hat{x},\hat{y})$ as follows: pick $\lambda\in[0,1]$ uniformly. For each v, let $\hat{x}_v=1$ iff $x^*_v\ge\lambda$, and $\hat{x}_v=0$ otherwise. Define $\hat{y}_e=\max(\hat{x}_u-\hat{x}_v,0)$ for each edge $e=(u,v)$. Using the same argument as in the proof of part 2 of Theorem 5.1 we can show that

$$\mathrm{E}\left[\alpha\sum_{v\in V}\hat{x}_v+\sum_{e\in E}\hat{y}_e\cdot c(e)\right] \;\le\; \alpha\sum_{v\in V}x^*_v+\sum_{e\in E}y^*_e\cdot c(e).$$

By optimality of (x^*,y^*), the inequality cannot be strict for any particular choice of λ, so $(\hat{x},\hat{y})$ is optimal with probability 1.

Second, observe that any integral solution $\hat{x}$ to the LP (2) (together with $\hat{y}$ defined by $\hat{y}_e=\max(\hat{x}_u-\hat{x}_v,0)$ for $e=(u,v)$) corresponds to an s-t cut in the graph G', obtained from G by adding an edge of capacity α from every vertex v to the sink t. The capacity of the cut is equal to the objective function value of (2). Hence, a minimum cut in G' gives us an integer optimal solution to the IP (2).

Since we want to solve (2) for varying α, we have a parametric flow problem on G'. Let S_α denote the s-side of the minimum cut in G' for a given value of α. Note that S_α minimizes $\alpha|S|+\gamma(S)$.

When $\alpha=0$, the minimum cut S_0 in G' is the same as the minimum cut in G. Hence, if $\gamma(S_0)>C$, then the MSBCC instance has no solution. We therefore assume that $\gamma(S_0)\le C$. As α tends to infinity, S_α shrinks, and for sufficiently large α, we will obtain $S_\alpha=\{s\}$. If $\gamma(\{s\})\le C$, then the MSBCC instance has a trivial optimal solution $(\{s\},V\setminus\{s\})$. We focus on the interesting case when $\gamma(\{s\})>C$.

Consider $\gamma(S_\alpha)$ as a function of α. It starts at a value $\gamma(S_0)\le C$ and eventually reaches a value that is greater than C. Hence there must be an α^* such that $\gamma(S_{\alpha^*-\epsilon})\le C$ and $\gamma(S_{\alpha^*+\epsilon})\ge C$ for arbitrarily small ϵ. For this value of α^*, there must be two min-cuts S^+ and S^- in G', such that $\gamma(S^-)\le C$ and $\gamma(S^+)\ge C$. Let x^+ and x^- be the characteristic vectors of S^+ and S^-, respectively. Since S^+ and S^- are both min-cuts in G', x^+ and x^- are optimal solutions of the IP (2) for parameter α^*. Hence, their convex combination

$x^* = \lambda \cdot x^+ + (1-\lambda) \cdot x^-$ is an optimal solution to the LP, together with y^* defined by $y_e^* = x_u^* - x_v^*$ for $e = (u,v)$. Choose λ such that $(1-\lambda) \cdot \gamma(S^-) + \lambda \cdot \gamma(S^+) = C$. We then obtain that

$$\begin{aligned}
\sum_{e \in E} y_e^* \cdot c(e) &= (1-\lambda) \sum_{e=(u,v)\in E} (x_u^- - x_v^-) \cdot c(e) + \lambda \sum_{e=(u,v)\in E} (x_u^+ - x_v^+) \cdot c(e) \\
&\le (1-\lambda) \cdot \gamma(S^-) + \lambda \cdot \gamma(S^+) \\
&= C
\end{aligned}$$

Hence x^* is also a feasible solution to the LP (1). Since the LP (2) is a Lagrangian relaxation of the LP (1), and (x^*, y^*) is optimal for (2), it must be also an optimal solution for (1).

6 Conclusion

We studied the problem of finding communities on the WWW. Considering that related pages have higher likelihood of linking to one another, one can hope to recognize communities using only the link structure. Then the problem becomes a graph-theoretic one of finding certain kinds of subgraphs in a graph. We considered three possible models of a graph community, one of which considers edge density, the other is based on inclusion and exclusion of edges, and the third combines the two.

In the process of considering algorithms for finding communities in a graph, we have identified a cleanly-defined graph-theoretic problem called Minimum size bounded capacity cut (MSBCC), which is both interesting by itself, and the solution to which would greatly help with finding subgraphs that are likely to be communities. The problem is NP-complete in general. We have given an exact algorithm for solving it in the special case of graphs with bounded treewidth. Also, we gave a bicriteria approximation result using the LP-relaxation of an integer programming formulation of the problem.

The main goal for future work is to obtain a single-criteria approximation for MSBCC. One example of a graph on which this may be hard is a source with one outgoing edge of capacity $k + \epsilon$, the other endpoint of which is part of a very large, very dense subgraph. There are no edges to the sink. The bound for the cut capacity is k. In this case, the optimal solution is to include the whole graph (except for the sink), but any technique that relaxes the capacity constraint will cut the only edge connected to the source.

Another possible direction of research is to try to derive lower bounds on the approximability of MSBCC.

References

[1] Y. Asahiro, K. Iwama, H. Tamaki, and T. Tokuyama. Greedily finding a dense subgraph. *Journal of Algorithms*, 34, 2000.

[2] U. Feige, G. Kortsarz, and D. Peleg. The dense k-subgraph problem. In *Proc. 25th ACM Symp. on Theory of Computing*, 1993.

[3] U. Feige and M. Seltser. On the densest k-subgraph problem. Technical report, The Weizmann Institute, Rehobot, 1997.

[4] G. Flake, S. Lawrence, C. L. Giles, and F. Coetzee. Self-organization of the web and identification of communities. *IEEE Computer*, 35, 2002.

[5] G. Flake, R. Tarjan, and K. Tsioutsiouliklis. Graph clustering techniques based on minimum cut trees. Technical Report 2002-06, NEC, Princeton, 2002.

[6] D. Gibson, J. Kleinberg, and P. Raghavan. Inferring web communities from link topology. In *Proc. 9th ACM Conf. on Hypertext and Hypermedia*, 1998.

[7] M. Girvan and M. Newman. Community structure in social and biological networks. *Proc. Natl. Acad. Sci. USA*, 99, 2002.

[8] M. Granovetter. The strength of weak ties. *American Journal of Sociology*, 78, 1973.

[9] David Kempe. 2nd CS685 reaction paper, 2002.

[10] J. Kleinberg. Authoritative sources in a hyperlinked environment. *J. of the ACM*, 46:604–632, 1999.

[11] R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins. Trawling the web for emerging cyber-communities. In *8th International World Wide Web Conference*, 1999.

[12] E. Lawler. *Combinatorial Optimization: Networks and Matroids.* Holt, Rinehard and Winston, 1976.

[13] R. Raz and S. Safra. A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP. In *Proc. 29th ACM Symp. on Theory of Computing*, 1997.

[14] Yahoo! www.yahoo.com.

A Approximation hardness for small α-communities

In this section, we restate and prove the approximation hardness result from [9] for the sake of completeness.

Theorem A.1 *1. It is NP-complete to decide whether a given graph G has an α-community of size at most k.*

2. Unless $P = NP$, the size of the smallest α-community cannot be approximated within a factor of $O(c \log n)$ in polynomial time, for some c.

3. The same results hold if we require to find a community of a specified vertex v.

Proof. Membership in NP is obvious. The NP-hardness will be implied by the approximation hardness, so we will only prove the second part of the Theorem (and show the few simple extensions necessary to prove the third). To avoid notational clutter, we assume here that $\alpha = \frac{1}{2}$.

We reduce from the SET COVER problem: Let $X = \{x_1, \ldots, x_n\}$ be the elements, and $S_1, \ldots, S_m \subseteq X$ the sets. The goal is to find a minimal number of sets S_i whose union covers X. To do so, we construct an undirected and unweighted graph G. Let X_i for an element x_i denote the set of all indices j such that $x_i \in S_j$.

We start with a large clique B of $4mn^2 + 1$ *bulk nodes*. These will be used solely to increase the degree of other nodes and force them to include many of their more "legitimate" neighbors.

We also have a set F of n *forcer nodes*, each of which is connected to n distinct bulk nodes. Forcer nodes are not connected among each other.

The other nodes are more important, and fall into two classes: first, there is an *element node* v_i for each element x_i, and the v_i form a clique. In addition, each v_i is connected to $2n + 1 - |X_i|$ distinct bulk nodes, and to each of the n forcer nodes.

For each set S_j, there are $2n + 1$ *set nodes* $s_{j,a}$, which are connected to form a clique (for fixed j). Each $s_{j,a}$ for $2 \le a \le 2n+1$ is connected to $2n$ distinct bulk nodes, whereas $s_{j,1}$ is connected to $2n + |S_j|$ distinct bulk nodes. Finally, there is an edge between $s_{j,1}$ and v_i whenever $x_i \in S_j$ in the set cover instance.

In the resulting graph, each element node v_i has degree exactly $4n$, each set node $s_{j,a}$ for $a \ge 2$ has degree $4n$, the nodes $s_{j,1}$ have degree $4n + 2|S_j|$, forcer nodes have degree $2n$, and the bulk nodes have degree at least $4mn^2$.

Given a set cover of size k, we can obtain a community as follows: Let C consist of all element nodes v_i, all forcer nodes, and all set nodes $s_{j,a}$ for which S_j is in the set cover. The size of C is $k(2n + 1) + 2n$. To verify that it is a community, notice that in the induced subgraph of C, each node $s_{j,a}$ for $a \ge 2$ has degree $2n$, and node $s_{j,1}$ has degree $2n + |S_j|$. Each node v_i has degree at least $2n$, because C contains all other $v_{i'}$, all forcer nodes, and a node $s_{j,1}$ for some set S_j containing x_i.

Conversely, consider a community C of size at most $k(2n + 1) + 2n$ for some $k < n$ (solutions with $k \ge n$ are rather uninteresting for set cover approximations). Because it is quite small, C cannot contain any bulk nodes — if it contained any one of them, it would have to contain at least $2mn^2 > m(2n+1)+2n$ of them to be a community.

Now, if C contains any set node $s_{j,a}$, it must contain all of the $s_{j,a'}$ for that particular j in order to be a community (otherwise, the degree requirement for $s_{j,a}$ would be violated, as none of its adjacent bulk nodes are included). By the size constraint, C contains the set node clique for at most k sets S_j, and we will prove that these sets S_j form a set cover.

In order to satisfy the degree requirement, all of the v_i adjacent to $s_{j,1} \in C$ must be included — in particular, any community C (of size at most $k(2n+1)+2n$) contains at least one element node v_i.

In order for the element node to have its required degree of $2n$ without bulk nodes, C must contain a forcer node, for C can contain at most n element nodes and $k \leq n-1$ nodes $s_{j,1}$. But in a community C with a forcer node and without bulk nodes, all element nodes v_i must be included to meet the forcer node's degree requirement. Thus, any community C without bulk nodes contains all element nodes.

Now, to meet an element node v_i's degree requirement of $2n$ without including bulk nodes, at least one set node adjacent to v_i must be included in C, for all forcer and element nodes would only yield degree $2n-1$. Hence, the sets S_j for which $s_{j,1} \in C$ indeed form a set cover of size at most k.

This completes the proof that the above reduction is approximation preserving, thus showing the inapproximability of the community problem, because Set Cover is known to be inapproximable within $O(c \log n)$ for some c (unless P = NP) [13].

When we ask about communities including a particular node v, we can use exactly the same reduction, and fix v to be a forcer node — the result is going to be the same, since forcer nodes were just shown to be included in all sufficiently small communities. ■