Math 197: Senior Thesis HARVEY MUDD Increasing Visibility of Vertices in Covert Networks Yaniv Ovadia

Introduction

Disrupting covert networks requires identifying and capturing key leaders. Recent work suggests that changing the topology of a covert network by vertex removal may cause a target vertex to participate in more communication and therefore become more visible.

In [5] Martonosi and Altner defined the *load* of a key vertex *k* in a simple undirected graph *G* to be

$$L(k,G) = \sum_{u,v\in G\setminus\{k\}} f_{u,v}(G) - f_{u,v}(G\setminus\{k\})$$
(1)

where $f_{u,v}(G)$ is the maximum flow (or equivalently, the minimum cut) between vertices u and v in the graph G.

We explored the feasibility of efficiently computing a subset of vertices whose removal causes the load on *k* to increase the most.



Figure 1: In this example graph, the load on *k* is only 3 because the flow between most pairs of vertices does not depend on *k*. If *v* is removed, the load increases to $(m+2)^2$.

The brute-force algorithm for computing the optimal single vertex to remove from *G* behaves as follows:

for $v \in V$ do

compute $f_{u,v}$ for all $u, v \in G \setminus \{v\}$

compute $f_{u,v}$ for all $u, v \in G \setminus \{v, k\}$

use results of the last two steps to calculate $L(k, G \setminus \{v\})$

end for

return the vertex *v* which caused the greatest load on *k*

Since this process involves many all-pairs max flow calculations, we employ Gomory-Hu tree construction algorithms (described in the next section). Furthermore, the all-pairs max flow calculations are often repeated on graphs which differ only by a single vertex deletion. Thus, we hoped to achieve a runtime improvement by designing an algorithm which uses results from an all-pairs max flow calculation in *G* to warm-start an all-pairs max flow calculation in $G \setminus \{v\}$.

The more general form of this problem asks for the subset *R* of a given set *S* of deletable vertices such that removing *R* from *G* maximizes the load on *k*. We proved this generalized problem to be NP-complete via a reduction from 3SAT.

Gomory-Hu Trees

A Gomory-Hu tree *T* for an undirected graph *G* is an encoding for the n - 1 unique minimum cuts in *G*. Specifically, for any two vertices $u, v \in G$, the value of a minimum (u, v)-cut (or equivalently a maximum (u, v)-flow) is equal to the weight of the lightest edge on the unique (u, v)-path in *T*. Furthermore, the partition for a cut of this size is obtained from the connected components that result when this lightest edge is removed from the tree.



Figure 2: An example graph and its corresponding Gomory-Hu tree.

Since our load metric depends on calculating the minimum cuts between every pair of vertices in a graph, the Gomory-Hu tree is particularly useful because it provides minimum cuts between all pairs of vertices, and can be constructed with only n - 1 maximum flow calculations (as opposed to $\binom{n}{2}$ via a näive algorithm). Gomory and Hu defined the first algorithm for constructing this structure in [3], and Gusfield later described a conceptually simpler algorithm in [4].

Non-Uniqueness of Gomory-Hu Trees

The Gomory-Hu tree for a graph is not unique, and different trees may result based on arbitrary decisions made in the construction process. Thus, in order to optimize the brute-force algorithm, it would be desirable to compute the specific tree which will change the least after a vertex deletion. We demonstrated that Gomory-Hu tree diversity results from the choice of minimum cuts (when more than one exists) as well as the order in which vertices are processed in the tree construction. This result suggests that constructing the specific tree which changes least after a vertex deletion is likely to be difficult.



Figure 3: An example of a graph with two distinct Gomory-Hu trees. The difference is due to choices in minimum cuts as opposed to vertex permutation.

Warm-Start Algorithm

Recall that the brute-force algorithm for finding the optimal single vertex to remove from the graph involves the construction of many Gomory-Hu trees for very similar graphs. To improve the efficiency of this process, we designed an algorithm which uses information from the construction of *G*'s Gomory-Hu tree to warm-start the construction of a Gomory-Hu tree for $G \setminus \{u\}$.

To update the Gomory-Hu tree *T*, for graph *G*, following a vertex deletion, we use Gusfield's algorithm to construct the new tree but warm-start each min cut computation within the algorithm using flow assignments acquired while calculating the existing Gomory-Hu tree.



Figure 4: Starting with an (a, b)-flow assignment (solid arcs), and a (b, c)-flow assignment (dashed arcs), we can construct an (a, c)-flow assignment.



Experimental Performance

We implemented both the brute-force algorithm, and the warm-start algorithm in Python, and tested their performance at searching for the load maximizing vertex removal on Barabási-Albert [1] and Erdős-Rényi [2] random graphs. The results in Figure 5 show that the warm-start algorithm is significantly faster on some graphs, particularly larger graphs with sufficient edge density.



Figure 5: Performance difference between brute-force and warm-start algorithms on Barabási-Albert and Erdős-Rényi random graphs.

References

- [1] A.L. Barabási and R. Albert. Emergence of scaling in random networks. *Science*, 286(5439):509, 1999.
- [2] P. Erdős and A. Rényi. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 5:17–61, 1960.
- [3] R.E. Gomory and T.C. Hu. Multi-terminal network flows. *Journal of the Society for Industrial and Applied Mathematics*, pages 551–570, 1961.
- [4] D. Gusfield. Very simple methods for all pairs network flow analysis. *SIAM Journal on Computing*, 19:143, 1990.
- [5] S.E. Martonosi, D.S. Altner, M. Ernst, and S. Plott. "Disrupting Terrorist Networks", Working Paper. 2009.