The Network Inhibition Problem

Patrick Cesarz, Gina Pomann, Luis de la Torre, Greta Villarosa

August 8, 2007

PROBLEM STATEMENT

Description

The electric power grid is a complex network that is particularly vulnerable to failure. In order to detect such vulnerabilities so that outages may be quickly corrected, it is necessary to investigate the Network Inhibition Problem. This problem seeks to maximally hinder the functioning of a network. The solution detects the smallest outage that will maximally inhibit the capability of a network. Thus, the Network Inhibition Problem allows us to find the greatest vulnerabilities present in such a complex network.

Since the Network Inhibition Problem is NP-complete, an alternative method that approximates the solution is the Inhibiting Bisection Problem[1]. In contrast to the Network Inhibition Problem, the Inhibiting Bisection Problem looks for a complete bisection of the network that maximally inhibits its function. Essentially, the problem seeks to minimize the number of lines cut in the network while maximizing the weight imbalance between the two sections of the network. The method for solving this problem involves optimizing one objective while pushing the other objective to a constraint. Unfortunately, the Inhibiting Bisection Problem is also NP-complete[1].

From here, however, we can evaluate a "Relaxed Version" of the Inhibiting Bisection Problem. Since we have two objectives (minimizing the cutsize *and* maximizing the imbalance), to satisfy both constraints, the relaxed version weighs the importance of each with respect to the other. Thus, we can find a bi-partitioning of the network that minimizes the function $\epsilon(cutsize) - (1 - \epsilon)(imbalance)$, where $0 < \epsilon < 1$, in which the choice of ϵ determines which objective is to be favored. So, the Inhibiting Bisection problem reduces to a type of maximum flow/minimum cut problem, which is solveable.

Procedure and Goals

We begin our research by first investigating the relaxed version of the Inhibiting Bisection Problem. As previously discussed, we are dealing with a type of maximum flow/minimum cut problem in which we seek to minimize the following function:

$$\epsilon|C(V_1, V_2)| - (1-\epsilon)|W(V_1 - W(V_2)|$$

Before gaining access to Ali Pinar's program in C++, we begin our preliminary work in Maple, which has its own minimum cut solver. Our Maple code will take in a given graph as input. Each graph will consist of a set of edges connected to a set of vertices with weights (W_i). Next, the program will set the edges between the vertices to a fixed ϵ and connect a source vertex to the generators and a terminal vertex to the consumers. The capacity of the edges connected to the source will be set to $(1-\epsilon)2W$, and those connected to the terminal will have capacities of $(1-\epsilon)(-2W)$. From here, the solver can calculate the minimum cut of a given graph for some ϵ.

After becoming familiarized with the relaxed version of the Inhibiting Bisection problem in Maple, we will then move on to investigating Ali Pinar's formulation of the relaxed Inhibiting Bisection Problem in C++. We will format his existing code and incorporate new code so that it will output our desired information. We will fix his code so that the program will solve the problem for every epsilon between 0 and 1 in increments of .01. For each epsilon, our program will store the cutsize, imbalance, edges, and partition associated with that epsilon and output this information in a LaTeX-formatted table. We will also construct the program so that it will return the maximum imbalance when prompted with a desired cutsize as input (or the minimum cut when prompted with a desired imbalance).

The next step is to extend this relaxed version of the problem to approximate the standard Inhibiting Bisection problem (with constraints).

For clarification, the Inhibiting Bisection Problem looks for the smallest bisection in graph that maximally inhibits the generator to consumer distribution. There are two objectives that can be satisfied for this problem [1]:

- imbalance constraint - minimize cutsize while keeping the imbalance is above a threshold

$$\text{minimize } |C(V_1, V_2)| \text{ subject to } |W(V_1) - W(V_2)| > B$$

- cut size constraint - maximize imbalance while keeping the cutsize is below a threshold

$$\text{maximize } |W(V_1) - W(V_2)| \text{ subject to } |C(V_1, V_2)| < D$$

We will write a program to vary ϵ in order to estimate the answer to this formulation of the inhibiting bisection problem. The program utilizes a binary search algorithm that estimates the optimal value of epsilon, i.e., the one that gives the maximum imbalance (minimum cutsize) for a given cutsize constraint (imbalance constraint). Essentially, the code designed for constraining cutsize will consist of a loop that decreases the value of epsilon if the size of the minimum cut is below the constraint or increases the value of epsilon if the cutsize exceeds the constraint. The code designed for constraining imbalance is similarly designed.

After we complete this code and have it working correctly, we will then test our approximations against the Integer Programming formulation of the Inhibiting Bisection problem [1] for small test problems. We know that the Integer Programming method scales poorly. So, if our approximations for the small example problems correlate well with the solutions from the Integer Programming formulation, we can assume that our program will give a good approximation of the solution for much larger graphs.

This research will help us find the value of ϵ which creates an ideal bisection of a given graph, i.e., one that maximizes the imbalance while minimizing the cut size. Ultimately, this information will help us find the most vulnerable areas in a network. Once we determine the best procedure for approximating the Inhibiting Bisection problem, we will be in a position to better evaluate the Network Inhibition problem.

References

[1] A. Pinar, Y. Fogel, and B. Lesieutre. The inhibiting bisection problem. Submitted to ACM 19th Symposium Parallel Algorithms and Architectures(SPAA) 2007.