Project Goal:

The goal of this project was to, given the components of a ship system, find the maximum power flow through each component in the system. The purpose of this is to then use the maximum flow values when designing components to ensure that they are capable of handling any operating conditions. In order to find this maximum flow, the ship system is modeled and analyzed as a directed graph.

Given Information:

In order to determine maximum flow through a system, my program started with an input file. This input file included a list of vertices representing ship components. For each vertex, the file also included the vertex number, the name of the component, its linearized efficiency as a decimal valued percentage, and all its adjacent vertices. Some vertices were labeled as sources or loads, in which case the file also included the rated power of said source or load. To ease the coding process, the program keeps track of vertices in terms of their vertex number, not the component name.

Finding Cycles:

The first step in this process was to find a combination of edges that, when removed, would break all cycles in the directed graph. In practice, the cycles first had to be tracked with a depth first search (DFS). The vertices and their adjacent vertices from the input file were used to create a directed graph, which was then traced in a DFS to find all cycles. While this was initially modeled as a directed graph, the cycle-tracing aspect was altered so that it traced a non-directed graph, where edges had paths in either direction. Cyclical graphs result in multiple paths in some sections of the graph. Breaking these cycles eliminates the potential of traversing a path that does not yield maximum flow. By modeling as a non-directed graph, the algorithm makes sure all these paths are taken away. It is also important to note that self-cycles, or cycles from one vertex to another and then immediately back to itself, are not recorded in this process.

Breaking Cycles:

After finding all the cycles in a given directed graph, the goal was then to find a combination of edges that broke all of these cycles. At first, the approach was to find all possible combinations of edges and see which were successful. This would take each combination, break all the edges in the combination, and re-run the DFS to track whether or not the combination successfully broke all cycles in the network. This was successful and accurate with small cases, but became very slow with larger cases. After analyzing some graph examples, some patterns were noticed in potential edge combinations. First, the number of edges in a successful combination must be the same as the number of cycles in the network. Second, when making a combination, all edges must have different lists of cycles that they are a part of. Lastly, every cycle must be included in the list of cycles an edge is included in for at least one edge in the combination.

For example, say there are three cycles in a network labeled cycle A, cycle B and cycle C, and vertices labeled numbers 1-9. If vertex 1 was in cycle A, vertex 2 in cycle B, and vertex 3 in cycle C, this would be a successful combination. Another example would be if vertex 1 was in cycles A, B, and C, vertex 2 in cycles A and B, and vertex 3 in cycles B and C. All cycle combinations are different, but still all cycles are represented. An example that would work would be 1: A, B, 2: A, 3: B. In this case, cycle C is not included and thus would not get broken.

In order to find edge combinations with these, I modeled the system as another DFS. With this, each edge was mapped to every other edge, and the DFS traced through all possible combinations, exiting the path when an edge didn’t satisfy the requirements. This then returned a list of edge combinations that would break all cycles in the system.

Maximum Flow:

Finding the overall maximum flow for each vertex involved iterating through all the possible edge combinations found to break the cycles and creating new graphs based on which edges were to be broken. The algorithm to find maximum flow through a given component is outlined in “Expanding the Design Space Explored by S3D”. This article explains maximizing flow for a component with only two connection nodes, where power can only flow either right to left or left to right. For components with more connection nodes, all combinations of nodes had to be assessed. This algorithm is further explained in “Determining Maximum Flow in a Network Given Any Possible Alignment” (unpublished). When calculating maximum flow, I first took whichever edge combination was in the current iteration and created a graph with those edges broken. I then used another DFS to find paths from each source to each vertex and from each vertex to each load. I used efficiency values given in the input file to find maximum flow values for each edge combination given the algorithm, then found the maximum for each component over all the edge combinations. Some components returned a zero value, which meant that there was only one node through which power flows in and out of the component.

Comments:

1. Add references section with proper citations
2. Technical writing, avoid “me” or “my”.  Passive voice is ok
3. When you do the writeup, pretend you knew what you were doing all along.  Only cite something that didn’t work if it provides helpful information.
4. Add images to help explain what is going on.

Outline:

Motivation

background

Problem statement

Approach

Examples

Conclusions

Future work

References