Max Flows and Min Cuts

Chan-Ching Hsu
cchsu@iastate.edu

February 10, 2016

1 Problem

A directed weighted (capacitated) graph $G = (V, E)$ consists of a set of nodes V and a set of directed edges E that connect them. An r/s cut C on a graph with two terminals is a partitioning of the nodes in the graph into two disjoint sets S and T such that the resource r is in S and the sink s is in T. The cost of a cut $C = S, T$ is defined as the sum of the costs of boundary edges (u, v) where $u \in S$ and $v \in T$. Cut cost is directed as it sums up weights of directed edges specifically from S to T. The minimum cut problem is to find a cut that has the minimum cost among all cuts. The minimum r/s cut problem can be solved by finding a maximum flow from r to s. The Ford-Fulkerson theorem states that a maximum flow saturates a set of edges in the graph dividing the nodes into S, T corresponding to a minimum cut. The algorithm is shown below.

1. for each edge $(u, v) \in E$ do
2. $\quad f(u, v) = f(v, u) = 0;$
3. while there is a path p in G_f do
4. $\quad c_f(p) = min c_f(u, v) | (u, v) \in p$
5. $\quad$ for each $(u, v) \in p$ do
6. $\quad\quad f(u, v) = f(u, v) + c_f(p)$
7. $\quad\quad f(u, v) = -f(u, v)$
8. $\quad$ end
9. end

2 Implementation

I ran everything on one of the servers, `research-6.ece.iastate.edu` [1], which is a Linux platform available to engineering students in ISU. We implemented the algorithm for searching for augmenting paths with the idea of Depth-first search (DFS) instead of Breadth-first search. A recursive implementation of DFS was considered to discover a path from the resource node to the sink node.

The pseudocode in the Wikipedia article on DFS [2] is able to provide with the basic idea of how our algorithm was developed. This is the base while we did not exactly following every step.
Those programs were developed with C [3] programming language. To run our algorithms on a network, a capacity matrix E, and vertices r and s would be given. E is taken to construct the flow network by assigning non-zero values and zero to the corresponding edges. The function of `Augmentation` returns a path P which has been found by iteratively performing the function `NextVertex` to discover and explore the neighboring nodes till s is reached or no path can be found. The path P is an ordered list of the vertices in an *rs-path*.
DFS algorithm also help us to extract the minimum cut when no augmenting path exists. C continues to be updated until there is no node can be reached by the exiting nodes in $\mathcal{R}$. The definition of C is equal to the $\mathcal{R}$ mentioned in class. Consequentially, the minimum capacity *minCut* can be obtained after we have the complete C. On the other hand, the maximum flow value *maxFlow* is updated as long as a new augmenting path is found. Throughout the implementation, it mostly challenged me whether the program was developed carefully and correctly. Starting from the first two simpler instances, I actually spent a lot of time on figuring out the conditions which should have been examined.
Here is our algorithm for finding an augmenting, as the procedure `Augmentation`:

```
procedure Augmentation(A, r):
    P = []; # store the augmenting path in order
    D = {}; # set of vertices have been explored when all neighbors have been visited
    P.append(r); # put resource node r into the first position of the path
    NextVertex(P, D, A, r, s); # call function to find next vertex
    if P is not None
        return P; # return P if path has been found

procedure NextVertex(P, D, A, u, s):
    for i in the network: # find the next node for u
        if A[u][i]>0: # edge (u,v) is accessible
            if i not in D and not in P: # next node must haven't been explored or in the path
                P.append(i); # put i into the path as the next vertex
                if s is reached: # the sink can be reached
                    return P; # the path is found
                else: # not reached s yet
                    if NextVertex(P, D, A, i, s) reaches s: # find the next node for i
                        break;
                    else: # s is not reachable from i
                        D.append(i); # return from i's neighbors all of which have been discovered
                        if i is the last node: # no more neighbors to be discovered
                            P.remove(u); # kick u off the path
            else if i is in P: # i is along the path somewhere
                if i is the last node: # no more nodes to discover
                    P.remove(u); # delete u from the path
            else if i is in D: # i has been explored
                continue; # consider node i+1
        else: # edge (u,v) is inaccessible anyway
            if i is the last node: # no other node to discover
                P.remove(u); # move out u from the path
```

Here is the algorithm for finding a minimum cut, as the procedure `Cut`:

```
procedure Cut(C, A, u):
    for i in the network: # make up the set of R
        if A[u][i] > 0: # flow of edge (u,v) is not used up
            if i not in C: # i is not in the set
                C.append(i); # add i into the set
                Cut(C, A, i); # find the cuts
```

Here is the main program calling functions to accomplish tasks, as the procedure `Main`:

```
procedure Main():
    E[u][v]; # the given capacity matrix from node u to v
    A[u][v]; # the remaining capacity on edge (u,v)
    A[u][v] = E[u][v] ; # the two matrices have identical values for each element initially
    r; # the resource node
    s; # the sink node
    C = {}; # the set of R
    maxFlow = 0; # the accumulated max flow
    flow = 0; # the flow value of the found path
    minCut = 0; # the minimum capacity

    do
        AP = Augmentation(A, r); # find the augmenting paths
        if AP is not None: # a path has been disclosed
            flow = 10000000;
            for edge(u,v) in AP: # check the minimum flow amongst all edges
                if flow > A[u][v]:
                    flow = A[u][v]; # update the available flow with the lower value
            maxFlow += flow; # increase the max flow value with the new number
            for edge(u,v) in AP: # update the residual graph
                A[u][v] -= flow;
                A[v][u] += flow;
    while AP is not None:

    C.append(r); # put r into the cut set
    Cut(C, A, u); # find cut

    for i in C:
        for j not in C:
            if A[i][j] = 0 && E[i][j] > 0: # capacity of edge (i,j) becomes 0
                minCut += E[i][j]; # increase minimum capacity value with new cut
```

All details can be supplied by request for the code of the implementation.

3 Solutions

We now discuss our solutions to the given problem instances.

3.1 Instance I1

Instance I1 is small enough that we were able to find the solution by hand. We used this as the very first case to verify our algorithm. The two in-class problems were also processed as input to

our algorithm to demonstrate the algorithm is able to work correctly in cases.

```
Maximum flow value: 17
Minimum cut capacity: 17
```

$\mathcal{R} = \{0, 2\}$, $\overline{\mathcal{R}} = \{1, 3, 4, 5, 6, 7\}$ with $\mid \mathcal{R} \mid = 2$, $\mid \overline{\mathcal{R}} \mid = 6$.

3.2 Instance I2

The instance although is a slightly larger case than the previous case, still, it doesn't introduce too much trouble for obtaining the a solution manually. To further validate the algorithm, the source and sink nodes in I1 and I2 were randomly chosen and the solutions were confirmed.

```
Maximum flow value: 33
Minimum cut capacity: 33
```

$\mathcal{R} = \{0, 1, 2, 6, 7, 8, 12, 13, 18, 19\}$, $\overline{\mathcal{R}} = \{3, 4, 5, 9, 10, 11, 14, 15, 16, 17, 20, 21, 22, 23\}$ with $\mid \mathcal{R} \mid = 10$, $\mid \overline{\mathcal{R}} \mid = 14$.

3.3 Instance I3

This instance provides a neat problem size to test our algorithm. We were able to discover a few minor defects corrected later by taking I3.

```
Maximum flow value: 29.1635
Minimum cut capacity: 29.1635
```

$\mathcal{R} = \{0, 2, 4, 5, 6, 7, 8, 9, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 22, 26, 28, 29, 30, 31, 34, 35, 36, 37, 38, 39, 40, 42, 43, 44, 45, 52, 53, 54, 60, 62, 63, 64, 66, 68, 69, 70, 72, 73, 74, 75, 77, 78, 79, 81, 84, 85, 88, 90, 93, 95, 96, 97, 99\}$,
$\overline{\mathcal{R}} = \{1, 3, 10, 21, 23, 24, 25, 27, 32, 33, 41, 46, 47, 48, 49, 50, 51, 55, 56, 57, 58, 59, 61, 65, 67, 71, 76, 80, 82, 83, 86, 87, 89, 91, 92, 94, 98\}$ with $\mid \mathcal{R} \mid = 63$, $\mid \overline{\mathcal{R}} \mid = 37$.

3.4 Instance I4

The last instance can be seen as a large case which requires apparently much more computational time to solve the problem as the size of the problem increases.

```
Maximum flow value: 507.2727
Minimum cut capacity: 507.2727
```

$\mathcal{R} = \{127\}$, $\overline{\mathcal{R}} = \mathcal{V} \setminus \mathcal{R}$ with $\mid \mathcal{R} \mid = 1$, $\mid \overline{\mathcal{R}} \mid = 499$.

References

[1] Linux Servers in College of Engineering, Iowa State University, `http://it.engineering.iastate.edu/remote/`

[2] Depth-first search, *Wikipedia*, `http://en.wikipedia.org/wiki/Depth-first_search)`

[3] C Programming Language, *Wikipedia*, `http://en.wikipedia.org/wiki/C_(programming_language)`