SIT320 - Advanced Algorithms

Credit Task 11: Network-based Algorithms

School of Information Technology, Deakin University

Overview of Learning (Module Summary)

GitHub Link
ChatGPT

Flow algorithms are used to solve problems that can be represented as a network of nodes and edges.
Networks can be represented by the use of directed graphs. In this module we were given two tasks to complete. The first task was to implement Karger's algorithm with the addition of weighted edges. The second was to implement the Ford-Fulkerson algorithm to find the maximum flow in a network. I will demostrate that finding the maximum flow in a network is equivalent to finding the minimum cut of a network. Where disconnection of a network is having two nodes that are not connected by any path.

Problem 1: Karger's Algorithm with Weighted Edges

Problem Statement

If we are presented with a network that includes weighted edges how would we modify Karger's algorithm to find the minimum cut of the network?
Karger's algorithm is a randomized algorithm that can be used to find the minimum cut of a network. The minimum cut is the smallest number of edges that need to be removed to disconnect the network.
If we are sucessful in finding the minimum cut, there will be a set S of edges(v, w) with v in P and w in P'.
The sum of the weights of the edges in S is the capacity of the cut.

Assumptions

We can assume that all edge weights are non-negative intergers and are chosen randomly from the range [1, 10].
The graph will be connected and not contain any self-loops.

Solution

I will be going off the asumption that **90% correct** is an acceptable accuracy for the algorithm.
This value was chosen as it is very close to one iteration(!) of the Kargers4 algorithm.
I ran the first two algorithms **(n * (n - 1) / 2 * log(1 / 0.1)) times** and returned the minimum cut found.

1. Karger's with random edge weights and contraction chosen probabilistically -> SLOW O(n^4) and ACCURATE with high number of iterations
2. Karger's with no edge weights and contraction chosen randomly -> SLOW O(n^4) and ACCURATE with high number of iterations
3. Recursive version of Karager's with the no edge weights -> FAST O(n^2 log^3 n) and ACCURATE with ONE iteration

I chose to implment a **probabilistic version** of Karger's algorithm with weighted edges as my solution to this task. I was originally puzzled about how weights could help us find the minimum cut of a network with Karger's algorithm. I was fixated on the idea that I **needed to know** with some certainty **where** the minimum cut was. Karger's finds edges to contract randomly and my solution to implement probabilities to chose an edge to contract is essentially doing the same thing.

The idea of Karger's algorithm is to randomly select an edge and merge the two vertices connected by the edge. This process is repeated until there are only two vertices left. The number of edges between the two vertices is the min-cut.

Pseudo-Code for Weighted Karger's Algorithm using probabilities to select an edge to contract:

```
Input: A weighted graph G = (V, E)
Output: A min-cut of G
While the number of vertices in the G is greater than 2:
    Create a list of edges -> edges
    Create a list of weights -> weights
    Create a list of probabilities -> probs
    Select an edge e in edges to contract, based on the probabilities in probs.
    Contract the edge e by merging the two nodes connected by e into a super-node.
    Add the super-node to the graph.
    Remove and self loops from the graph if they exist.
    Remove the original nodes that were contracted.
Return the min cut, which is the number of edges between the two remaining nodes.
```

Key Take aways

I believe that the idea of this task was a way to lead us to the Ford-Fulkerson algorithm. Adding weights to Karger's algorithm did not lead to any insights, but rather allowed me to appriciate the ideas presented in the next task. I feel as if we posses the knowledge of where the minimum cut is, we can manipulate the weights of the edges closest and find a solution more efficiently. There have been improvements in Karger's algorithm, and I attempted to implement these improvments. *see Kargers4* in the notebook.

Randomised Algorithms

Karger's is an example of a **Monte Carlo** algorithm. It is a randomized algorithm in the sense that it uses randomisation to decide what to do next. To find a solution with any certainty, the algorithm must be run **multiple times**. The algorithm is **probabilistic** as it has a chance of returning an incorrect solution. Randomised algorithms present a facinating idea that we can use randomness to find efficient *good-enough* solutions to difficult problems.

Problem 2: Ford-Fulkerson Algorithm

Problem Statement

Prove that finding the maximum flow in a network is equivalent to finding the minimum cut of a network.
The Ford-Fulkerson algorithm is a greedy algorithm that can be used to find the maximum flow in a network.

The maximum flow is the maximum amount of flow that can be pushed through the network from the source to the sink given the capacity of the edges.

Solution

My solution to this problem was to first implment Graph and Node classes. These laid the foundation for the Ford-Fulkerson algorithm. I created *edges*, *weights* and *flow* dictionaries in the Graph class. This class also had methods to update the flow and find paths. The Node class simply contained a *data* field and a *neighbors* list. I also found it useful to create a specific Residual Graph class. The reason for this was that it was awkward to reverse the edges of the original graph. This class also contained a method to update the weights in the residual graph.
The main algorithm was comprised of four helper methods; *is_reachable*, *get_path*, *increase_flow* and *print_min_cut*.
I also crated a *Ford_Fulkerson* function that called these helper methods.

The steps to find the min_cut and prove that it is equivalent to the max_flow are as follows:

```
1. Create a graph and its residual graph
2. Call the Ford_Fulkerson function
    2a. Set the max_flow to 0
    2b. While there is a path from source to sink
        2b1. Find the path
    2c. Set the flow to the minimum of the weights in residual graph from the path
    2d. Update the flow in the graph
    2e. Update the weights in the residual graph
    2f. Add the flow to the max_flow
    2g. Return the max_flow
3. Find all nodes reachable from the source in the residual graph
    3a. Add the nodes to a visited set
    3b. Add the neighbors of the nodes to a queue
4. Set min_cut to 0
    4a. Loop through the visited set
    4b. Loop through the neighbors of the node in the residual graph
    4c. If the neighbor is not in the visited set
    4d. Add the weight of the edge in the residual graph between the neighbor and the node to the min_cut
5. Print the min_cut and max_flow
```

Key Take aways

The min-cut in a network is equal to the max flow. This is because the max flow is the maximum amount given a capacity that can be pushed through the network from the source to the sink. While the min-cut is the minimum number of edges that need to be removed to disconnect the network. The min-cut is the sum of the weights of the edges that are removed, and the max flow is the sum of the weights of the edges that are not removed. The max-flow/min-cut has use cases such as organising transportation networks and distributing oil through refinery pipelines.

Idea behind the Ford-Fulkerson Algorithm

Let P be a path from source to sink in a flow network G. We need to satisfy the following conditions:

- For each properly oriented edge (u,v) in P, we have Flow(u,v) < Capacity(u,v)
- For each properly oriented edge (u, v) in P, we have Flow(u,v) > 0

Let delta be the minimum of the differences Capacity(u,v) - Flow(u,v) over all **properly oriented** edges (u,v) in P.

```
                | Flow(u, v),            if (u, v) is not in P
Flow*(u, v) =   | Flow(u, v) + delta,    if (u, v) is in P and properly oriented
                | Flow(u, v) - delta,    if (u, v) is in P and improperly oriented
```

Since the edge (u,v) in P is increased by delta, the value of Flow*(u,v) is delta greater than the value of Flow(u,v) before the augmentation.

Readings and Resources

Discrete Mathemathics - Richard Johnsonbaugh

- Chapter 10: Network Models

Introduction to Algorithms - Cormen, Leiserson, Rivest & Stein

- Chapter 26: Maximum Flow

A New Approach to the Minimum Cut Problem - Karger & Stein 1996

More randomized algorithms: Karger’s MinCut Algorithm - Aleks Ignjatovic (School of Computer Science and Engineering University of New South Wales)