1 Flow Network Basics

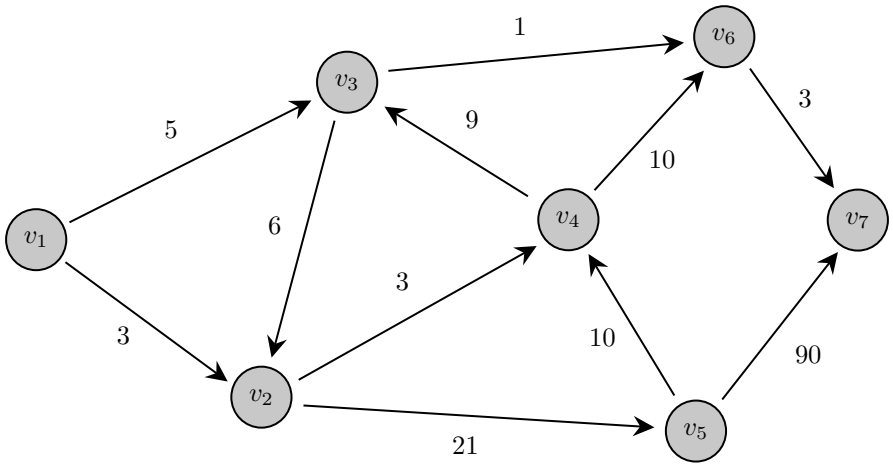
1.1 Flow Networks and Flow

A flow network is a directed graph $G = (V, E)$, in which each edge $(u, v) \in E$ has a non-negative capacity $c(u, v) \geq 0$. We will assume that there are no self-loops, and also that if $(u, v) \in E$ then $(v, u) \notin E$. There is one vertex designated as the source, and one vertex designated as the sink.

A flow in G is a real-valued function $f : V \times V \to \mathbb{R}$ that satisfies the capacity constraint property and the flow conservation property i.e., the flow from one vertex to another must be non-negative and must not exceed the given capacity, and the total flow into a vertex other than the source or sink must equal the total flow out of that vertex.

That is quite a mathematical and formal definition is there an intuitive answer?

Of course! Imagine we have the below graph where each edge represents a pipe and each vertex represents some connection between these pipes and the capacity of each edge is simply how much water we can get through the pipe. In this way a flow is exactly as it sounds i.e., some flow of water through the pipes.



$c(v_2,\ v_5) = 21$

1.2 Brief History of Max Flow

One of the most natural questions to ask is:
How much water can we move through the pipes?

Well two mathematicians - T. E. Harris and F. S. Ross - asked the same question in 1954, but rather on the capacity of Soviet train networks. This may not have been and was likely not the first formulation of the Maximum flow problem, but this was the first official formulation of the problem in their paper "Fundamentals of a Method for Evaluating Rail Net Capacities". Writing "Given a railway network with known or estimated capacities ... establish a traffic pattern in the network which will enable the maximum amount to be moved" (figure 1). The Maximum flow problem was solved by Ford and Fulkerson in the following year.

1.3 Maximum Flow

The value $|f|$ of a flow f can be defined as the total flow out of the source minus the flow into the source and the maximum flow is the maximum this value can be with a flow satisfying the previous conditions.

1.4 Residual Graph and Augmenting Paths

A residual graph is the resulting graph from the available capacities of a flow network. In a flow network graph, a flow from u to v would be one edge with flow over the capacity. In a residual graph, there are two edges, one from u to v representing the flow and an edge in the opposite direction, which would be from v to u, representing the residual capacity, which is the total capacity subtracting the existing flow.

Augmenting paths are then paths in this residual graph that the algorithm modifies to increase the total maximum flow from the source to the sink by increasing the flow through the specific path that has been found out. If there is a path from the source to sink vertices in a residual graph, it represents that there is capacity for more flow. Thus, the algorithm augments this existing path to the point where there is no more flow that is able to go down that specific path.

2 Edmonds-Karp Algorithm

2.1 How Does It Work?

In Jack Edmonds and Richard M. Karp's paper published in 1972 titled, *Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems*, they present their proposed algorithm.

As it is an optimisation on the Ford-Fulkerson algorithm, it works similarly in augmenting paths to get to a maximum flow. However different from Ford-Fulkerson, it specifies to use a breath-first search as a way of finding paths to be augmented. The use of a breath-first search graph transversal guarantees that the augmented paths found will be the shortest ones, thus saving time from the original algorithm.

The time complexity is then $O(VE^2)$, where V is the number of vertices and E is the number of edges, which differs from the Ford-Fulkerson Algorithm, which has a time complexity of $O(|f|E)$. The difference in time is due to BFS being able to guarantee finding the shortest path opposed to Ford-Fulkerson's non-specific approach.

2.2 Implementation

Below is an implementation of the Edmonds-Karp Algorithm in javascript using an vertex array and edge matrix representation of a graph

```
function EdmondsKarpResult(graph) {
    let maxFlow = 0;

    // call the bfs() method and straight up backtrack and augment
    while (true) {
        graph.bfs();

        // check if the end was reached
        if (!graph.vertices[1].bfsSeen) break;

        // now backtrack through the bfs
        let pathVertices = [];
        let pathEdges = [];

        let minCapacity = Infinity;
        let backtracker = graph.vertices[1];
        pathVertices.push(backtracker);
        while (true) {
            if (backtracker.bfsPred != null) {
                let edge = graph.edges[backtracker.bfsPred][backtracker.id];
                pathEdges.push(edge);
                minCapacity = Math.min(minCapacity, edge.capacity);

                // traverse the backtracker
                backtracker = graph.vertices[backtracker.bfsPred];
                pathVertices.push(backtracker);
            } else {
                break;
            }
        }

        for (let v of pathVertices) {
```

```
            v.bfsInPath = true;
        }

        for (let e of pathEdges) {
            e.bfsInPath = true;
        }

        graph.publish();

        // augment the flow
        for (let e of pathEdges) {
            e.capacity -= minCapacity;
            graph.edges[e.toVertex][e.fromVertex].capacity += minCapacity;
        }
        // Increment the current maxFlow
        maxFlow += minCapacity;
    }

    return maxFlow;
}
```

Listing 1: Code for Edmonds-Karp

3 Dinic's Algorithm

3.1 How Does It Work?

Yefim Dinitz wrote and published his own algorithm, the Dinitz's algorithm, which became unknown to the non-Russian population. Due to modifications made by Shimon Even and Alon Itai, the algorithm became known as Dinic's algorithm.

Dinic's algorithm is also an optimisation on the Ford-Fulkerson and Edmond-Karp algorithm. The difference is that Dinic's uses a level graph, which determines the level from the source vertex. Through this, the algorithm can send multiple flows into the graph which is known as the blocking flow. It uses the levels to determine the paths quicker by determining if there is a path from a level 0 to 1 to 2... to the sink vertex's level. If there is no more flow that can be added on, meaning there is no capacity or path from the residual graph then it has reached blocking flow. This use of level graph means that flow is more efficiently sent and the program will terminate earlier than the previous iterations.

The time complexity is $O(EV^2)$, where E is the number of edges and V is the number of vertices. This difference in time occurs to Dinic's ability to find an augmenting path in $O(VE)$ time. This is done $O(V)$ iterations thus resulting in a total $O(EV^2)$ time.

3.2 Implementation

Below is an implementation of Dinic's Algorithm in javascript using an vertex array and edge matrix representation of a graph

```
function DinicsResult(graph) {

    let maxFlow = 0;

    // call the bfs() method and then do a dfs through the level graph
    while (true) {
        graph.bfs();

        // first check if the end was reached
        if (!graph.vertices[1].bfsSeen) break;

        while (true) {
            // perform a dfs on the level graph
            graph.dfs();

            // check if the sink was reached
            if (!graph.vertices[1].dfsSeen) break;

            // backtrack through the dfs
            let pathVertices = [];
            let pathEdges = [];
```

```
22
23              let minCapacity = Infinity;
24              let backtracker = graph.vertices[1];
25              while (true) {
26                  if (backtracker.dfsPred != null) {
27                      let edge = graph.edges[backtracker.dfsPred][backtracker.id];
28                      pathEdges.push(edge);
29                      minCapacity = Math.min(minCapacity, edge.capacity);
30
31                      // traverse the backtracker
32                      backtracker = graph.vertices[backtracker.dfsPred];
33                      pathVertices.push(backtracker);
34                  } else {
35                      break;
36                  }
37              }
38
39              for (let v of pathVertices) {
40                  v.dfsInPath = true;
41              }
42
43              for (let e of pathEdges) {
44                  e.dfsInPath = true;
45              }
46
47              // augment the flow
48              for (let e of pathEdges) {
49                  e.capacity -= minCapacity;
50                  graph.edges[e.toVertex][e.fromVertex].capacity += minCapacity;
51              }
52              // increment the maxFlow calculated so far
53              maxFlow += minCapacity;
54          }
55      }
56
57      return maxFlow;
58  }
```

Listing 2: Code for Dinic's

4 Applications

4.1 Extensions and Applications of Maximum Flow Problem

Using the maximum flow problem, we can apply this to other concepts and thus use this relatively efficient algorithm to determine solutions to more complex and harder computational questions. These extended problems link to others in theoretical programming and thus are important to understand.

One of the most common extensions of the maximum flow problem is edge-disjoint paths, where the maximum flow is the number of edge-disjoint paths. An edge-disjoint path is a set of paths that edges do not overlap in another path or the same one. Similarly, we can use maximum flow to consider vertex-disjoint paths whilst also modifying the maximum flow problem to consider vertex capacities as well. This increases the range of problems that we can create and solve for that have many real world applications and thus fundamentally understanding of the maximum flow problem and how to solve it can aid in many more difficult problems.

Further applications of the maximum flow problem include bipartite matching and tuple selection which focus on splitting the graph into different categories in order to enable for specific matching for real life problems and are particularly used in machine learning algorithms and many other common theoretical programming problems.

Thus through learning about the basics of the maximum flow problem there are countless other problems and issues that can be reduced to maximum flow which further how we consider different computational problems.

4.2 Real-world Applications

Why should we care? Flow networks are used in a variety of places such as electricity networks, water distribution and sewage networks, transport networks, finance, image segmentation, data security, multi-camera scene reconstruction, distributed computing, and more. Clearly, flow networks have a broad range of extremely important applications which allow our modern world to function, this being said, flow networks and models based on flow networks have their limitations when it comes to their real world applications.

For example, for flow network models of traffic and airlines to be accurate, accurate data must be used as input, however, unforeseen circumstances are inevitable and data cannot always be accurate. In these situations, consequences can be catastrophic and we are left with traffic jams, flight delays, insufficient water access, and crashes.

Flow networks can be used to model transport networks and can be utilised to optimise traffic such as how to improve upon infrastructure and network design, considering noise pollution and traffic control due to modeling real-time traffic. Network flow algorithms, can be used to determine the capacity of road infrastructure systems and how they will respond to the introduction of new roads and increased flow requirements (where traffic bottlenecks may form) as well as where redundancies can be introduced to improve reliability. Flow network models of transport infrastructure are not perfect however, considerations of psychology and even fluid dynamics must be used to more accurately model and predict traffic patterns not just applying network flow models.

Electrical networks require distributing electricity and doing it efficiently to where it is required. This can be on a small scale, such as something like a circuit or a house, or networks of larger scales such as power grids. Flow networks allow us to estimate the capacity of electrical grids and determine whether they have the redundancy to reliably withstand daily peaks with unforeseen breaks or sudden changes to power needs, they allow us to model and plan for the effects of large scale outages.

Water is another utility that can benefit from considering flow network optimisation. Crucial infrastructure such as sewers and fresh water pipelines as well as irrigation systems rely on maximising and optimizing flow and distribution to where it is needed. If water has to flow greater distances, then there will often be larger associated capital and infrastructure costs due to extra materials for the manufacturing and construction of the physical pipelines as well as the energy requirements of water transport.

Flow networks can be used to model financial systems. Allowing the modelling of loss transmission from borrowers to intermediaries and to lenders, and allowing a better understanding of the effects of shocks to different parts of the global financial system and different markets e.g., how a collapse of Thai currency and Thai banks can lead to the collapse of the Korean Won and Korean banks. There are also limitations, however, unknown lines of transmission can be unaccounted for and lead to inaccurate models.

5 Appendix

ESTIMATION OF NETWORK CAPACITIES

The U.S. Army FM 101-10 (Staff Officers Field Manual)[6] sets forth, as a sort of rule of thumb for planning purposes, the capacities of single and double track lines. It lists these capacities as 10 and 30 trains each way per day in a theater of operations and sets forth conditions that will affect these capacities. These data are derived from operating experience, primarily in the Second World War in Europe. They are adequate for the purpose for which they are intended, and are also useful as an aid in evaluating the capacities of entire rail nets.

This latter problem may be stated roughly as follows:

Given, a railway network with known or estimated capacities for the individual lines, with certain locations designated as sources or ''origins'' of supplies and certain other locations designated as ''destinations.''

Required, to establish a traffic pattern in the network which will enable the maximum amount to be moved, on a sustained basis, from the origins to the destinations, and to determine the value of this maximum.

Figure 1: Statement of Maximum Flow Problem in "Fundamentals of a Method for Evaluating Rail Net Capacities" by Harris and Ross

6 References

1. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, 2003, CLRS Introduction to Algorithms
2. Abd-Alsabour, N. 2020, 'The Maximum Flow Problem', International Journal of Engineering Research and Technology
3. Woods, A. J. 2017, Applications of Flow Network Models in Finance
4. Kolmogorov, V. and Zabih, R. 2003, Multi camera Scene Reconstruction via Graph Cuts
5. Harris, T. E. Ross, F. S. 1955, Fundamentals of a Method for Evaluating Rail Net Capacities
6. Edmonds, J Karp, R. M. 1972, 'Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems', Journal of the ACM

7. Dinitz, Y. 2006, 'Dinitz' algorithm: the original version and Even's version', Theoretical Computer Science: essays in Memory of Shimon Even