Ford-Fulkerson Maximum Flow

Analysis of Augmenting Paths Techniques

Pedro Rodriguez
University of Colorado at Boulder
p.rodriguez@colorado.edu

ABSTRACT

Abstract text. Abstract text. Abstract text. Abstract text. Abstract text. Abstract text. Abstract text.

1. INTRODUCTION

The first research into network flow algorithms began in the early and mid 20th century. Initial research by A.N. Tolstoi sought to optimize production networks such as railroads in the Soviet Union[1]. The subject later picked up interest amongst the American scientists Ford and Fulkerson who devised the first known algorithm for computing the maximum flow through a network[2].

The maximum flow problem is defined by: Finding the maximum flow through a network from a source vertex s to a sink vertex t in a directed graph G which has edges labeled with flow capacities.

Twenty years after Ford-Fulkerson published their results Edmonds and Karp independently improved upon the algorithm by replacing the depth-first search strategy used in Fork-Fulkerson with breadth-first search[3]. This change improved the algorithmic efficiency from $O(E|f|)$ where f is the maximum flow to $O(VE^2)$. This idea was further improved upon by Dinic's who exploited the property that the flow paths were monotonically increasing to achieve an even better $O(V^2E)$ running time[4]. All of these algorithms share a common strategy: find an augmenting path from s to t to push flow across and repeat until there is no such path. At this point the algorithm terminates and returns the maximum flow value and paths.

Beyond the natural applications of maximum flow algorithms to optimizing supply chains, routing, and others it has gained popularity as a sub-algorithm call in more advanced problems. For example, it was recently used to solve the k-clique densest subgraph problem[5], and various computer vision such as object category segmentation, image deconvolution, super resolution, texture restoration, character completion, and 3D segmentation[6]. It has also been used to solve other problems such as minimum path cover in directed acyclic graphs, maximum cardinality bipartite matching, and in the "real-world" baseball elimination, airline scheduling, and circulation-demand[7].

2. PROBLEM STATEMENT

2.1 Formal Problem Statement

Now we proceed to formally defining the maximum flow problem. The problem is defined as given a graph with:

1. Vertex set V
2. Directed edge set E
3. Source vertex $s \in V$ such that no edges are incident
4. Sink vertex $t \in V$ such that no edges leave it
5. Edge capacities $c_{uv} \in \mathbb{R}^+, \forall (u,v) \in E$

And subject to the conditions:

1. $f_{uv} \coloneqq$ variable indicating flow from u to v
2. $f_{uv} \leq c_{uv}, \forall (u,v) \in E$
3. $\forall v \in V$ such that $v \neq s,t : \sum_{(u,v)\in E} f_{uv} = \sum_{(v,u)\in E} f_{vu}$

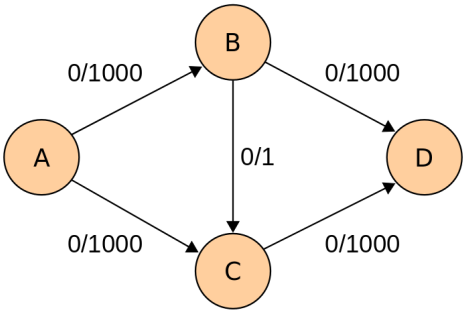
Find the flow assignments f_{uv} such that network flow $|f| \coloneqq \sum_{(s,v)\in E} f_{sv}$ is maximized.

2.2 Graphical Depiction

To fix ideas we will continue by describing a concrete instance of the problem. For example, consider the example in figure 1[8]. In the figure, each edge is labeled by a pair of numbers which are presented as f_{uv}/c_{uv}. This represents how much flow f_{uv} is passing through an edge with capacity c_{uv}. By inspection, it is easy to deduce that the maximum flow is 2000 with 1000 flowing through the top and 1000 flowing through the bottom. This representation makes the meaning of the conditions (1) and (2) clear. Condition (3) expresses that the flow into any vertex which is not a source or sink must be the flow out of that vertex. Finally, the objective can be explained by noting (with proof later) that the maximum flow from A to D is equivalent to taking a cut across A.

Before moving on, lets consider a simple algorithm: find a path through the network and push the maximum possible flow through until pushing flow is impossible. One can see this may result in the optimal solution, but if one path found is the one cross BC, then the optimal flow will never be found. This hints that the simple graph construction is insufficient to solve the maximum flow problem.

Figure 1: Simple flow network



3. ALGORITHM ANALYSIS

In this section we will first present the very similar Ford-Fulkerson and Edmonds-Karp algorithms, prove their correctness, and analyze their asymptotic behavior in both time and space complexity.

3.1 Algorithm

Before proceeding to the algorithm itself, we must make a few additional definitions and constructions. First we will define a $G_f(V, E_f)$ which we will call the residual graph. In this graph, the capacities are defined as $c^f_{uv} = c_{uv} - f_{uv}$ and there is zero flow. We additionally define that $\forall(u, v) \in Ef_{uv} = -f_{vu}$. Combining these two loosens the constraints from the original graph to allow cases where if $f_{uv} > 0$ and $c_{uv} = 0$ that $c^f_{vu} = c_{vu} - f_{vu} = f_{uv} > 0$. As will be proven, this allows for algorithm to function correctly. We now proceed to stating the main algorithm as Algorithm 3.1.

Algorithm 1 Ford-Fulkerson and Edmonds-Karp

```
1: function MAX_FLOW(G,C,S,T)
2:     Input: G, c, s, t as described in 2.1
3:     Output: Flow f_max and paths to achieve it
4:     loop
5:         path ← augmenting_path(G_f, s, t)
6:         if path = None then
7:             break
8:         end if
9:         f_min = min_{(u,v)∈path} c^f_uv
10:        for (u, v) ∈ path do
11:            f_uv ← f_uv + f_min
12:            f_vu ← f_vu − f_min
13:        end for
14:    end loop
15:    f_max ← Σ_{(s,v)∈E} f_sv
16:    paths ← paths/flows from s to t in created graph
17:    return f_max, paths
18: end function
```

The algorithm proceeds in two alternating phases: finding an augmenting path from s to t in G_f and pushing the maximal viable flow across that path. When an augmenting path cannot be found, then the algorithm terminates and returns the current maximal flow.

We define an augmenting path as any path from s to t in G_f such that $c^f_{uv} > 0\ \forall(u, v) \in path$. The algorithm was not previously specified because its definition determines whether the algorithm is known as Ford-Fulkerson or Edmonds-Karp. If the augmenting path is found using depth first search then the algorithm is called Fork-Fulkerson. If the augmenting path is found using breadth first search then the algorithm is called Edmonds-Karp. With that in mind, we now define the two possible augmenting path algorithms. Note that the $0th$ element of a list is considered the back and the last element of a list is considered the front. Additionally the notation $vec[a; b]$ is short for creating a vector of length b with values initialized to a.

```
1: function AUGMENTING_PATH(G_f, S, T, SEARCH)
2:     V, E_f ← G_f
3:     search_list ← list()
4:     if search = BFS then
5:         pop ← list.pop_front
6:         push ← list.push_back
7:     else
8:         pop ← list.pop_front
9:         push ← list.push_front
10:    end if
11:    push(search_list, s)
12:    distances ← vec[∞; |V|]
13:    parents ← vec[∞; |V|]
14:    u ← pop(search_list)
15:    while u ≠ None and u ≠ t do
16:        for v in neighbors(u) do
17:            if distances[v] = ∞ and c^f_uv > 0 then
18:                distances[v] ← distances[u] + 1
19:                parents[v] ← u
20:                push(search_list, v)
21:            end if
22:        end for
23:        u ← pop(search_list)
24:    end while
25:    return u == t, reconstruct_path(s, t, parents)
26: end function

27: function RECONSTRUCT_PATH(S, T, PARENTS)
28:    path ← list()
29:    node ← t
30:    loop
31:        path.append(node)
32:        if node == s then
33:            break
34:        end if
35:        path ← parents[node]
36:    end loop
37:    return reverse(path)
38: end function
```

4. REFERENCES

[1] Alexander Schrijver. On the history of the transportation and maximum flow problems. *Math.*

Program. (), 91(3):437–445, 2002.

[2] L R Ford and D R Fulkerson. Maximal flow through a network. *Canadian journal of Mathematics*, 1956.

[3] Jack Edmonds and Richard M Karp. Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems. *J. ACM ()*, 19(2):248–264, 1972.

[4] E A Dinic. Algorithm for solution of a problem of maximum flow in a network with power estimation, Soviet Math. Doll. 11 (5), 1277-1280,(1970).

[5] Charalampos Tsourakakis. The k-clique densest subgraph problem. In *Proceedings of the 24th international conference on world wide web*, pages 1122–1132. International World Wide Web Conferences Steering Committee, 2015.

[6] Tanmay Verma and Dhruv Batra. MaxFlow Revisited: An Empirical Comparison of Maxflow Algorithms for Dense Vision Problems. *BMVC*, pages 1–12, 2012.

[7] Wikipedia. Maximum flow problem, 2016. [Online; accessed 20-April-2016].

[8] Wikipedia. Fork-fulkerson algorithm, 2016. [Online; accessed 20-April-2016].