Ford–Fulkerson algorithm

The Ford–Fulkerson method (named for L. R. Ford, Jr. and D. R. Fulkerson) is an algorithm which computes the maximum flow in a flow network. It was published in 1956. The name "Ford–Fulkerson" is often also used for the Edmonds–Karp algorithm, which is a specialization of Ford–Fulkerson.

The idea behind the algorithm is simple. As long as there is a path from the source (start node) to the sink (end node), with available capacity on all edges in the path, we send flow along one of these paths. Then we find another path, and so on. A path with available capacity is called an augmenting path.

Algorithm

Let be a graph, and for each edge from to , let be the capacity and be the flow.

We want to find the maximum flow from the source to the sink . After every step in the algorithm the following is maintained:

- Capacity constraints: for all edges.
- Skew symmetry: for all edges.
- Flow conservation: for all nodes except the source and the sink.

This means that the flow through the network is a legal flow after each round in the algorithm. We define the residual network to be the network with capacity and no flow.

When no more paths in step 2 can be found, will not be able to reach in the residual network. If is the set of nodes reachable by in the residual network, then the total capacity in the original network of edges from to the remainder of is on the one hand equal to the total flow we found from to , and on the other hand serves as an upper bound for all such flows. This proves that the flow we found is maximal. See also Max-flow Min-cut theorem.

If the graph has multi Sources and Sinks, we act as follows. Suppose that and . Add a new source with an edge from to every node , capacity . And add a new sink with an edge from to every edge,
node \( t \in T \), with capacity \( f(t^*, t) = d_t \sum_{(v, t) \in E} f(v, t) \). Then applying the Ford–Fulkerson algorithm. Also if every nodes \( u \) has constraint \( d_u \), we replace this node with two nodes \( u_{in}, u_{out} \), and an edge \((u_{in}, u_{out})\) with capacity \( f(u_{in}, u_{out}) = d_u \) and then applying the Ford–Fulkerson algorithm.

**Complexity**

By adding the flow augmenting path to the flow already established in the graph, the maximum flow will be reached when no more flow augmenting paths can be found in the graph. However, there is no certainty that this situation will ever be reached, so the best that can be guaranteed is that the answer will be correct if the algorithm terminates. In the case that the algorithm runs forever, the flow might not even converge towards the maximum flow. However, this situation only occurs with irrational flow values. When the capacities are integers, the runtime of Ford-Fulkerson is bounded by \( O(Ef) \) (see big O notation), where \( E \) is the number of edges in the graph and \( f \) is the maximum flow in the graph. This is because each augmenting path can be found in \( O(E) \) time and increases the flow by an integer amount which is at least 1.

A variation of the Ford–Fulkerson algorithm with guaranteed termination and a runtime independent of the maximum flow value is the Edmonds–Karp algorithm, which runs in \( O(VE^2) \) time.

**Integral example**

The following example shows the first steps of Ford–Fulkerson in a flow network with 4 nodes, source \( A \) and sink \( D \). This example shows the worst-case behaviour of the algorithm. In each step, only a flow of 1 is sent across the network. If breadth-first-search were used instead, only two steps would be needed.

<table>
<thead>
<tr>
<th>Path</th>
<th>Capacity</th>
<th>Resulting flow network</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial flow network</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( A, B, C, D ) ( \min(c_f(A, B), c_f(B, C), c_f(C, D)) = \min(c(A, B) - f(A, B), c(B, C) - f(B, C), c(C, D) - f(C, D)) = \min(1000 - 0, 1 - 0, 1000 - 0) = 1 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Final flow network</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( A, C, B, D ) ( \min(c_f(A, C), c_f(C, B), c_f(B, D)) = \min(c(A, C) - f(A, C), c(C, B) - f(C, B), c(B, D) - f(B, D)) = \min(1000 - 0, 0 - (-1), 1000 - 0) = 1 )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Notice how flow is "pushed back" from \( C \) to \( B \) when finding the path \( A, C, B, D \).
Non-terminating example

Consider the flow network shown on the right, with source $s$, sink $t$, capacities of edges $e_1$, $e_2$ and $e_3$ respectively 1, $r = (\sqrt{5} - 1)/2$ and 1 and the capacity of all other edges some integer $M \geq 2$. The constant $r$ was chosen so, that $r^2 = 1 - r$.

We use augmenting paths according to the following table, where $p_1 = \{s, v_4, v_3, v_2, v_1, t\}$, $p_2 = \{s, v_2, v_3, v_4, t\}$ and $p_3 = \{s, v_1, v_2, v_3, t\}$.

<table>
<thead>
<tr>
<th>Step</th>
<th>Augmenting path</th>
<th>Sent flow</th>
<th>Residual capacities</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>$r^0 = 1$</td>
<td>$r$ 1</td>
</tr>
<tr>
<td>1</td>
<td>${s, v_2, v_3, t}$</td>
<td>1</td>
<td>$r^0$ $r^1$ 0</td>
</tr>
<tr>
<td>2</td>
<td>$p_1$</td>
<td>$r^1$</td>
<td>$r^2$ 0 $r^1$</td>
</tr>
<tr>
<td>3</td>
<td>$p_2$</td>
<td>$r^1$</td>
<td>$r^2$ $r^3$ 0</td>
</tr>
<tr>
<td>4</td>
<td>$p_1$</td>
<td>$r^2$</td>
<td>0 $r^3$ $r^2$</td>
</tr>
<tr>
<td>5</td>
<td>$p_3$</td>
<td>$r^2$</td>
<td>$r^2$ $r^3$ 0</td>
</tr>
</tbody>
</table>

Note that after step 1 as well as after step 5, the residual capacities of edges $e_1$, $e_2$ and $e_3$ are in the form $r^n$, $r^{n+1}$ and 0, respectively, for some $n \in \mathbb{N}$. This means that we can use augmenting paths $p_1$, $p_2$, $p_3$ and $p_3$ infinitely many times and residual capacities of these edges will always be in the same form. Total flow in the network after step 5 is $1 + 2(1 + r^2)$. If we continue to use augmenting paths as above, the total flow converges to $1 + 2 \sum_{n=1}^{\infty} r^n = 3 + 2r$, while the maximum flow is $2M + 1$. In this case, the algorithm never terminates and the flow doesn't even converge to the maximum flow.

Python implementation

```python
class Edge(object):
    def __init__(self, u, v, w):
        self.source = u
        self.sink = v
        self.capacity = w
    def __repr__(self):
        return "%s->%s:%s" % (self.source, self.sink, self.capacity)

class FlowNetwork(object):
    def __init__(self):
        self.adj = {}
        self.flow = {}

    def add_vertex(self, vertex):
        self.adj[vertex] = []

    def get_edges(self, self, v):
```
def add_edge(self, u, v, w=0):
    if u == v:
        raise ValueError("u == v")
    edge = Edge(u,v,w)
    redge = Edge(v,u,0)
    edge.redge = redge  #redge is not defined in Edge class
    redge.redge = edge
    self.adj[u].append(edge)
    self.adj[v].append(redge)
    self.flow[edge] = 0
    self.flow[redge] = 0

def find_path(self, source, sink, path, path_set):
    if source == sink:
        return path
    for edge in self.get_edges(source):
        residual = edge.capacity - self.flow[edge]
        if residual > 0 and not (edge,residual) in path_set:
            path_set.add((edge, residual))
            result = self.find_path(edge.sink, sink, path + [(edge,residual)], path_set)
            if result != None:
                return result

def max_flow(self, source, sink):
    path = self.find_path(source, sink, [], set())
    while path != None:
        flow = min(res for edge,res in path)
        for edge,res in path:
            self.flow[edge] += flow
            self.flow[edge.redge] -= flow
        path = self.find_path(source, sink, [], set())
    return sum(self.flow[edge] for edge in self.get_edges(source))

Usage example
For the example flow network in maximum flow problem we do the following:

```python
>>> g = FlowNetwork()
>>> [g.add_vertex(v) for v in "sopqrt"]
[None, None, None, None, None, None]
>>> g.add_edge('s','o',3)
>>> g.add_edge('s','p',3)
>>> g.add_edge('o','p',2)
>>> g.add_edge('o','q',3)
>>> g.add_edge('p','r',2)
>>> g.add_edge('r','t',3)
```
>>> g.add_edge('q','r',4)
>>> g.add_edge('q','t',2)
>>> print g.max_flow('s','t')
5

Notes

References


External links

- Another Java animation (http://www.cs.pitt.edu/~kirk/cs1501/animations/Network.html)

Media related to Ford–Fulkerson algorithm at Wikimedia Commons
Article Sources and Contributors


Image Sources, Licenses and Contributors


License

Creative Commons Attribution–Share Alike 3.0
//creativecommons.org/licenses/by-sa/3.0/