The MPM Algorithm for Maximum Flow

Ruijie Fang

The purpose of this note is to give, in expository form, a description to a simple and elegant algorithm by Malhotra, Pramodh-Kumar, and Maheshwari in 1978 for solving max-flow in $\Theta(n^3)$-time on a general graph G of n vertices and m edges. Faster and unlike Dinitz's algorithm or Ford-Fulkerson variants, it requires no involved analysis and nor is complicated in implementation.

This note is taken and organized from *The Design and Analysis of Algorithms* by Dexter Kozen.

Preliminaries

A path flow of a flow graph is a flow that only has values along a simple path.

Definition (Path flow)**.** A path flow of a flow graph $G = (V, E, c)$ is a flow f with nonzero values only on a unique simple path from s to t; given a path flow f, there exists a number d and a length-$k + 1$ simple path $u_0u_1...u_{k-1}u_k$ with $s = u_0, t = u_k$ such that

$$\begin{aligned} f(u_i, u_{i+1}) &= d, \\ f(u_{i+1}, u_i) &= -d \text{ for } 0 \leq i \leq k-1 \\ f(u, v) &= 0 \text{ for all other } (u, v). \end{aligned}$$

A level graph L_G is a directed BFS tree of G with root s and with back edges as well as edge between levels deleted.

We say that a flow is saturated when, the value of the flow on an edge of the graph equals the capacity of that edge. This leads us to the idea of a blocking flow:

Definition (Blocking flow)**.** A blocking flow f is a flow such that every $s - t$ path in the level graph L_G contains at least one saturated edge.

Of course, only looking at edges is no good (this is where the m-factors in the running time come from), so MPM requires us to look at vertices; to do that, we need to define vertex capacities:

$$c(v) = \min \left\{ \underbrace{\sum_{u \in V} c(u, v)}_{\text{total capacity in}} , \underbrace{\sum_{u \in V} c(v, u)}_{\text{total capacity out}} \right\}$$

The MPM Algorithm

We proceed in phases: for the start of each phase, we compute a residual graph and its associated level graph L_G. During each phase, if t is not in L_G, there is no st-path in the residual graph, and we terminate. Otherwise, we further prune L_G by deleting all vertices that are not on a st-path:

Procedure Do-MPM-Phase(G: level graph) {

1. Let $v := \arg\min_{v \in V} c(v); d := \min_{v \in V} c(v)$; (Find a vertex of min capacity)
2. If $d = 0$, delete v and all incident edges; update capacity of neighboring vertices. Go back to 1. Otherwise, go to step 3:

3. Push d units of flow from $v - t$ and pull d units of flow from s into v (overall, walk a $s - v - t$ path).
   (a) Push to sink $(v - t)$: Saturate, in order of growing capacities, the outgoing edges from v. By construction, at most one edge is partially saturated, with more edges possibly untouched and some fully saturated. Delete all fully saturated edges. Propagate this process onto each affected neighboring vertex, and keep going until we reach t. The procedure terminates and is correct since by definition, d is the minimum viable flow to push through all vertices on a $v - t$ path, since it is the minimum vertex capacity of the residual graph.
   (b) Pull from source $(s-v)$: Works analogously. Saturate, in order of growing capacities, the incoming edges into v. By construction, at most one edge is partially saturated, with more edges possibly untouched and some fully saturated. Delete all fully saturated edges. Repeat the process on each vertex from which flow is taken during saturation until the procedure propagates back to s. It is similarly, always possible to pull d units of flow from s, since d is the global minium vertex capacity, and all edges on this $s - v$ path either have capacity equal d or greater than d.
4. Now, either all incoming edges or all outgoing edges of v have been saturated and deleted (since either all incoming edge capacities or all outgoing edge capacities sum up to d by construction). So v becomes useless and is no more on any $s - t$ paths. We delete v and its incident edges from the level network and update neighbor capacities. Go back to step 1.

}

We repeat `Do-MPM-Phase`, which runs in $O(n^2 + m)$-time, in a loop, computing the residual graph and feeding it into the function. We can use something like a Fibonacci heap to maintain the current v, d pairs in $O(n \log n)$-overall amortized time. There are n phases, so the overall running time is $O(n^3 + mn) = O(n^3)$.