Assignment 2: MPM Algorithm in Network Flow

Subhrajyoty Roy (BS - 1613)

February 4, 2019

1 Introduction

The MPM Algorithm is a modification of the Dinic's Algorithm, which proposes a certain way to select the **Blocking Flows** in the graph which is used to augment the current flow. At the same time, this algorithm also prunes off the vertices and edges which are guaranteed to be never used again. This ingenious suggestion is due to Malhotra, Pramodh-Kumar and Maheshwari.

We shall start by defining the following terms first;

1. **In-potential of a vertex:** The in-potential of a vertex v is defined as the sum of the residual capacities of the incoming edges to v in the residual layered graph L_f. In mathematical terms,
$$\rho_f^+(v) = \sum_{e:e=(u,v)} c_f(e)$$
where $c_f(e)$ denotes the residual capacity of edge e in the residual graph L_f.

2. **Out-potential of a vertex:** The out-potential of a vertex v is defined as the sum of the residual capacities of the outgoing edges from v in the residual layered graph L_f. In mathematical terms,
$$\rho_f^-(v) = \sum_{e:e=(v,u)} c_f(e)$$

3. **Potential of a vertex:** The potential of a vertex v is the minimum of the in-potential and the out-potential of that vertex. Intuitively, potential means the amount of flow that can be pushed through that vertex in the residual graph, G_f. In mathematical terms,
$$\rho_f(v) = \min\left\{\rho_f^+(v), \rho_f^-(v)\right\}$$

4. **Reference node and Reference Potential:** A reference vertex is a vertex which has minimum potential among all available vertices. If the current vertex set is V, then reference node is $r = \arg\min_v \rho_f(v)$. The reference potential is the minimum potential, i.e. the potential of reference node r i.e. $\rho_f(r)$.

2 Algorithm and Subroutines

The MPM algorithm has been described in Algorithm 1.

Observe that, in the above algorithm, we use two subroutines which is not mentioned before, namely **Push** and **Pull** subroutine. The push subroutine namely pushes the flow from a specified vertex to the sink vertex, and the pull subroutine takes the same amount of flow from source vertex to the specified vertex.

Result: Output a maximum flow f of graph G
Initialize $f(e) = 0$ for all edges e in graph G;
while *source s and sink t are connected in Residual graph G_f* **do**
    Compute the Layered graph L_f;
    while *source s and sink t are still conneceted in Layered graph L_f* **do**
        Compute potential for each vertex in L_f;
        Remove all vertices which are not on any path from s to t, and also remove corresponding edges incident to them (These are the vertices having either no incoming or no outgoing edges in L_f);
        Find the reference vertex r and m be the reference potential;
        Let U be the set of all vertices u such that $\rho_f(u) = m$, i.e. U is the set of all possible reference vertices;
        Push m unit of flow from r to sink t and accordingly update residual capacities;
        Pull m unit of flow from source s to r accordingly update residual capacities;
        Delete all vertices u in U, and remove its incident edges from L_f;
    end
end
output current flow f as maximum flow;

Algorithm 1: MPM Algorithm

Input: flow amount m, reference vertex r, sink t
if *v is not equal to the sink vertex t* **then**
    Take the edges that are outgoing from r in some order, and saturate each one with flow, unless and until saturating one more would lift the total flow used over m;
    Assign all remaining flow to the next outgoing edge (not necessarily saturating it), so the total outflow from v becomes exactly m;
    foreach *endpoint u of fullly or partially saturated edges e* **do**
        Augment the flow for this edge e to the original network G;
        Compute the total incoming flow in u, let this be $f^+(u)$;
        Recursively call Push($f^+(u)$, u, t);
    end
    Delete all saturated edges from the layered graph L_f;
end

Algorithm 2: Push

Input: flow amount m, reference vertex r, source t
if *v is not equal to the sink vertex t* **then**
  Take the edges that are incoming to r in some order, and saturate each one with flow, unless and until saturating one more would lift the total flow used over m;
  Assign all remaining flow to the next incoming edge (not necessarily saturating it), so the total outflow from v becomes exactly m;
  foreach *other endpoint u of fullly or partially saturated edges e* **do**
    Augment the flow for this edge e to the original network G;
    Compute the total outgoing flow from u, let this be $f^-(u)$;
    Recursively call Push($f^-(u)$, u, s);
  end
  Delete all saturated edges from the layered graph L_f;
end

Algorithm 3: Pull

3 Proof of Correctness and Complexity Analysis

Consider the terminating condition of the MPM Algorithm. Observe that, if s and t are disconnected in the residual graph G_f, then the current flow f is a maximum one, as there will be no path from s to t in the residual graph G_f. Therefore, to show the proof of correctness of MPM Algorithm, we can just show that this algorithm will terminate, and it always keeps a valid feasible flow.

Lemma 3.1. *In one complete iteration of the inner while loop, the MPM algorithm outputs a valid feasible flow in L_f.*

Proof. To be a valid feasible flow, we require the checking of the following two constraints;

1. Flow at every edge is upper bounded by its capacity.

2. The incoming total flow at any vertex is equal to the total outgoing flow from that vertex.

For the first constraint, we observe that, the push and pull subroutine always dynamically updates the residual capacity of each edge (Consider the first line in the foreach loop for Push and Pull subroutine). As we always consider the layered graph L_f and never pushes more than the capacity of each edge, the first constraint is always satisfied.

For the second constraint for the vertex v, we consider two cases.

1. If vertex v is a reference vertex in the current iteration, then MPM algorithm pushes m units of flow from v and pulls the same amount of flow from v, where m is the reference potential. Observe that, after this iteration, the reference vertex v, as is saturated, gets deleted and hence the second constraint is satisfied for vertex v.

2. If vertex v is not a reference vertex in the current iteration, then there are two subcases;

   (a) Vertex v does not appear in the path from s to t via the current reference vertex r. In this case, during the Push or Pull subroutine, the vertex v is never visited, hence its incoming flow and outgoing flow remains same.

   (b) Vertex v appears in the path from s to t via the current reference vertex r. In this case, some amount of flow is either pulled or pushed through v during the call of those

subroutines. However, in such cases, the push and pull is called with the amount of flow same as incoming or outgoing total flow from v. (Note the last line in the foreach loop).

This shows that for these vertices also, the second constraint is satisfied.

□

Lemma 3.2. *During one iteration of the outer while loop, the inner while loop computes a Blocking flow in layered graph L_f, i.e. when the inner while loop breaks, it outputs a blocking flow in L_f.*

Proof. Suppose not, i.e. after one complete iteration of the outer while loop we get a flow g in L_f (it is indeed a feasible flow by previous lemma), which is not a blocking flow.

If this is assumed, then by the definition of Blocking flow g, there does not exist any path P from source s to sink t, such that for all edges e on the path, $g(e) < c(e)$, i.e. the flow on each edge on that path is strictly lesser than the capacity of that. However, that means, due to the updation of the flowing in L_f, the source s and sink t are still connected in L_f, which implies that the inner while loop has not been broken by now, and another iteration of the inner loop is still possible. This contradicts that we are considering one complete iteration of the outer loop. □

Theorem 3.1. *If the algorithm terminates, it terminates with a feasible flow.*

Proof. In each iteration of the outer while loop, a blocking flow on L_f is computed (by previous lemma). Now, capacities of edges in G_f do not exceed the capacities of the edges in original graph (for forward edges) and do not exceed the amount of flow pushed through that edges in the original graph (for backward edges).

The blocking flow in L_f satisfy the capacity constraint where the capacities the residual capacities, as L_f is a subgraph of G_f. Hence, augmenting the current flow with this blocking flow g always results in a feasible flow satisfying the capacity constraint. □

Lemma 3.3. *The algorithm performs one iteration of the outer while loop in $2|V|^2+|E|$ updation operations and in at most $|V|$ operations of finding maximum potential vertex along with the computation of potentials.*

Proof. Let us consider the i-th iteration of the inner while loop. In this case, we process at most $|V| - 1$ other nodes, and also the reference node. Also, assume that we process E_i many edges and make them saturated. These saturated edges are deleted from the layered graph and never visited again. Now, note that, whenever a push or pull operation is performed on a vertex, there can be at most one partially saturated edge for that vertex, which may be visited again during later iterations. Therefore, the total number of partially saturated edges is upper bounded by $|V|$. Therefore, we require $2|V| + E_i$ many updating operations to be performed at i-th iteration.

Finally, we have $|V|$ iterations in total, as we are deleting the reference vertex (i.e. at most one vertex) from the layered graph at each iteration, hence, the total number of updation of flows on edges is bounded by $\sum_i 2|V| + E_i \leq 2|V|^2 + |E|$.

Now, in $|V|$ iterations, we require that many operations to find the reference vertices in each iteration, however, computing the reference potentials require checking the capacity of each edge (if done naiively) for each iteration, thereby introducing a factor $|V||E|$ in one iteration of the outer loop. However, it can be avoided, if we compute the potential of each vertex naively only once per outer iteration, and as we call the push and pull subroutine to augment the flow, it is possible to update the potentials also by the same cost of updation of flows. □

Lemma 3.4. *Let $d(f)$ denote the shortest distance between s and t in the current residual graph G_f. Clearly, $d(f) \in \{1, 2, \ldots |V| - 1, \infty\}$. $d(f)$ increases by at least one in each iteration of the outer loop.*

Proof. Let g be the blocking flow in L_f computed by inner while loop. Let L_{f+g} be the layered graph after augmenting the flow f with flow g. And let G_{f+g} is the residual graph after augmentation. Then there are 3 types of edges e in L_{f+g},

1. $e \in L_f$, this type of edges are forward edges that advances by only one layer in the layered graph. These edges does not contribute to the increment of $d(f)$.

2. $e \in G_f$ and $e \in L_f$. In this case, clearly, as $e \in G_{f+g}$ as edge e is part of layered graph L_{f+g} by assumption. Hence, e is a newly created edge after augmentation. Now, the newly created edge must be because of augmentation of some edges in L_f, which implies that the newly created edge must be in backward direction of L_f.

3. $e \in G_f$ and $e \notin L_f$. Also, as e is not in L_f, hence it is not a forward edge in G_f, but a backward edge from a layer to its previous layer.

Let us consider a shortest path P in L_{f+g}. By above discussion, each edge e is either of type 1, 2 or 3. Hence, $d(f + g) \geq d(f)$. Now, the only way $d(f + g) = d(f)$ can happen, if and only if each edge e on the path P is of type 1. But, in that case, $e \in L_f$ and $e \in L_{f+g}$ for all $e \in P$ for any path P from source to sink. Since, $e \in L_f$,

$$f(e) < c(e)$$

and since $e \in L_{f+g}$,

$$f(e) + g(e) < c(e)$$

as L_{f+g} only contains the forward edges, hence the augmented flow g is only added. From the last inequality, we get; $g(e) < (c(e) - f(e))$ and from the first inequality; $(c(e) - f(e)) > 0$. In this case, with respect to the path P, each edge on this path has some positive residual capacity as $(c(e) - f(e)) > 0$ and the blocking flow value $g(e)$ is strictly less than the residual capacity $(c(e) - f(e))$ for each edge on this path. This contradicts the fact that g is a blocking flow in L_f. Hence, we must have $d(f + g) > d(f)$ which proves the assertion. □

Theorem 3.2. *The MPM Algorithm terminates in $O(|V|^3)$ time.*

Proof. Since $d(f)$ increases by one in each iteration of the outer loop, there can be at most $|V| - 1$ iterations before $d(f) = \infty$ which means the source s and sink t gets disconnected. Once the source and the sink are disconnected, we know that the algorithm terminates and outputs the current flow.

In each iteration, we require about $2|V|^2 + |E|$ many updations for flow as seen in lemma 3.3, while we require about $|V|$ many operations of finding maximum potential vertex, which requires $|V|^2$ many computations. Also, as described in the proof of lemma 3.3, we require $|E|$ computation to initialize the potentials of each vertex, and we can maintain it in $O(|V|^2)$ as described for each iteration of outer while loop. Therefore, each iteration of the outer while loop can be bounded by $O(|V|^2)$ many simple computational steps (like updating flows and updating potentials).

Therefore, the MPM algorithm takes $|V| \times O(|V|^2)$ i.e. $O(|V|^3)$ many steps. □

A direct corollary of theorem 3.1 and theorem 3.2 is that they together imply that the MPM algorithm terminates with a feasible flow, and clearly, in the residual graph corresponding to this feasible flow, the source and sink are disconnected, implying that the output flow is indeed maximum.