1 Max flow - Min Cut (continued)

1.1 Flow Decomposition

In many situations we would like, given a feasible flow, to know how to *recreate* this flow with a bunch of $s-t$ paths. In general a feasible flow might also contain cycles. This motivates the following definition.

A *primitive element* is either:

1. A simple path P from s to t with flow δ.
2. A cycle Γ with flow δ.

Theorem 1 (Flow Decomposition Theorem)**.** *Any feasible flow f can be decomposed into no more than m primitive elements.*

Proof. The proof is by construction; that is, we will give an algorithm to find the primitive elements. Let $A_f = \{(i,j) \in E : f_{ij} > 0\}$, and $G' = (V, A_f)$.

Pseudocode

While there is an arc in A_f adjacent to s (i.e. while there is a positive flow from s to t), do:
Begin DFS in G_f from s, and stop when we either reach t along a simple path P, or found a simple cycle Γ. Let ξ denote the set of arcs of the primitive element found.
Set $\delta = \min_{(i,j)\in\xi}\{f_{ij}\}$.
Record either (P, δ) or (Γ, δ) as a primitive element.
Update the flow in G by setting: $f_{ij} \leftarrow f_{ij} - \delta \ \forall (i,j) \in \xi$.
Update A_f. Note that the number of arcs in A_f is reduced by at least one arc.

If no arc in A_f leaves s, then A_f must consist of cycles. While $A_f \neq \emptyset$ do:
Pick any arc $(i,j) \in A_f$ and start DFS from i.
When we come back to i (which is guaranteed by flow conservation), we found another primitive element, a cycle Γ.
Set $\delta = \min_{(i,j)\in\Gamma}\{f_{ij}\}$.
Record (Γ, δ) as a primitive element.
Update the flow in G by setting: $f_{ij} \leftarrow f_{ij} - \delta \ \forall (i,j) \in \xi$.
Update A_f. Note that the number of arcs in A_f

Note that the above algorithm is correct since, every time we update the flow, the new flow must still be a feasible flow.

Since, each time we found a primitive element we removed at least one arc from A_f, we can find at most m primitive elements. □

1.2 Dinic's algorithm for maximum flow

Dinic's algorithm is also known as the Blocking Flow Algorithm, or shortest augmenting paths algorithm. In order to describe this algorithm we need the following definitions.

Layered Network: Given a flow, f, and the corresponding residual graph, $G_f = (V, A_f)$, a *layered network $AN(f)$* (AN stands for Auxiliary Network) is constructed as follows:
Using breadth-first-search in G_f, we construct a tree a rooted at s. Each node in V gets a distance label which is its distance in the BFS tree from s. Thus, the nodes that have an

incoming arc from s get assigned a label of 1 and nodes at the next level of the tree get a label of 2, etc. Nodes with the same label are said to be at the same layer. Let the distance label of t be $k = d(t)$ then all nodes in layer $\geq k$ are removed from the layered network and all arcs are removed except those which go from a node at one layer to a node at the next consecutive layer.

Note that this layered network comprises all of the shortest paths from s to t in G_f.

Blocking flow: A flow is called blocking flow for network G' if every s, t path intersects with at least one saturated arc. Note that a blocking flow is not necessarily a maximum flow!

Stage k: We call stage k the augmentations along paths in the layered network where $k = d(t)$ (a network that includes k layers).

Now we are ready to give the basic implementation of Dinic's algorithm.

Dinic's algorithm

$f_{ij} = 0 \ \forall (i,j) \in A$
Use BFS to construct the first layered network $AN(f)$.
While $d(t) < \infty$ do:
    Find a blocking flow f' in $AN(f)$.
    Construct G_f and its respective $AN(f)$.
End While.

Find a blocking flow (naive implementation)

While DFS finds a path P from s to t do:
    Augment the flow along P.
    Remove saturated arcs in $AN(f)$.
    Cross out nodes with zero indegree or outdegree equal to zero.
End While.

Theorem 2. *Dinic's algorithm solves correctly the max-flow problem.*

Proof. The algorithm finishes when s and t are disconnected in residual graph and so there is no augmenting path. Thus by the augmenting path theorem the flow found by Dinic's algorithm is a maximum flow. □

Complexity Analysis

We first argue that, after finding the blocking flow at the k-layered network, the shortest path in the (new) residual graph is of length $\geq k + 1$ (i.e. the level of t must strictly increase from one stage to the next one). This is true since all paths of length k intersect at least one saturated arc. Therefore, all the paths from s to t in the next layered network must have at least $k + 1$ arcs (all paths with length $\leq k$ are already blocked). Thus, Dinic's algorithm has at most n stages (one stage for each layered network.

To find the blocking flow in stage k we do the following work: 1) Each DFS and update takes $O(k)$ time (Note that by definition of $AN(f)$ we can't have cycles.), and 2) we find the blocking flow in at most m updates (since each time we remove at least one arc from the layered network).

We conclude that Dinic's algorithm complexity is $O(\sum_1^n mk) = O(mn^2)$.

Now we will show that we can improve this complexity by finding more efficiently the blocking flow. In order to show this, we need to define one more concept.

Throughput: The *throughput* of a node is the minimum of the sum of incoming arcs capacities and the outgoing arcs capacity (that is, $thru(v) \triangleq \min\left\{\sum_{(i,v)\in A^f} u^f_{v,j}, \sum_{(v,f)\in A^f} u^f_{v,f}\right\}$). The throughput of a node is the largest amount of flow that could possibly be routed through the node.

The idea upon which the improved procedure to find a blocking flow is based is the following. We find the node v with the minimum throughput $thru(v)$. We know that at this node we can "pull" $thru(v)$ units of flow from s, and "push" $thru(v)$ units of flow to t. After performing this pulling and pushing, we can remove at least one node from the layered network.

Find a blocking flow (improved)

While there is a node $v \neq s, t$ with $thru(v) > 0$ in $AN(f)$.
  Let v be the node in $AN(f)$ for which $thru(v)$ is the smallest; Set $\delta \leftarrow thru(v)$.
  Push δ from v to t.
  Pull δ from s to v.
  Delete (from $AN(f)$) all saturated arcs, v, and all arcs incident to v;
End While

Push δ from v to t

$Q \leftarrow \{v\}$; $excess(v) \leftarrow \delta$
While $Q \neq \emptyset$
  $i \leftarrow$ first node in Q (remove i from Q)
  For each arc (i, j), and while $excess(i) > 0$
    $\Delta \leftarrow \min\left\{u^f_{ij}, excess(i)\right\}$
    $excess(i) \leftarrow excess(i) - \Delta$
    $excess(j) \leftarrow excess(j) + \Delta$
    $thru(j) \leftarrow thru(j) - \Delta$
    $f_{ij} \leftarrow f_{ij} + \Delta$
    if $j \notin Q$ add j to end of Q
  End For
End While

The "Pull" procedure has a similar implementation.

Theorem 3. *Dinic's algorithm solves the max-flow problem for a network $G = (V, A)$ in $O(n^3)$ arithmetic operations.*

Proof. As before, the algorithm has $O(n)$ stages.

At each stage we need to create the layered network with BFS. This takes $O(m)$ complexity.

The operations at each stage can be either saturating push along an arc or an unsaturating push along an arc. So we can write number of operations as sum of saturating and unsaturating pull and push steps, $N = N_s + N_u$. We first note that once an arc is saturated, it is deleted from graph, therefore $N_s = O(m)$. However we may have many unsaturating steps for the same arc.

The key observation is that we have at most n executions of the push and pull procedures (along a path) at each stage. This is so because each such execution results in the deletion of the node v with the lowest throughput where this execution started. Furthermore, in each execution of the push and pull procedures, we have at most n unsaturated pushes – one for each node along the path. Thus, we have at each stage at most $O(n^2)$ unsaturated pushes and $N = N_s + N_u = O(m) + O(n^2) = O(n^2)$. Therefore, the complexity per stage is $O(m + n^2) = O(n^2)$.

We conclude that the total complexity is thus $O(n^3)$. □

1.2.1 Dinic's algorithm for unit capacity networks

Definition 4. *A unit capacity network is an $s-t$ network, where all arc capacities are equal to* 1.

Some problems that can be solved by finding a maximum flow in unit capacity networks are the following.

- Maximum Bipartite matching,
- Maximum number of edge disjoint Paths.

Lemma 5. *In a unit capacity network with distance from s to t greater or equal to ℓ the maximum flow value $|\hat{f}|$ satisfies $\sqrt{|\hat{f}|} \leq 2\frac{|V|}{\ell}$.*

Proof. Construct a layered network with V_i the set of the nodes in Layer i (See Figure 1). And let $S_i = V_0 \cup V_1 \cup \cdots \cup V_i$
$1 \leq i \leq \ell - 1$.

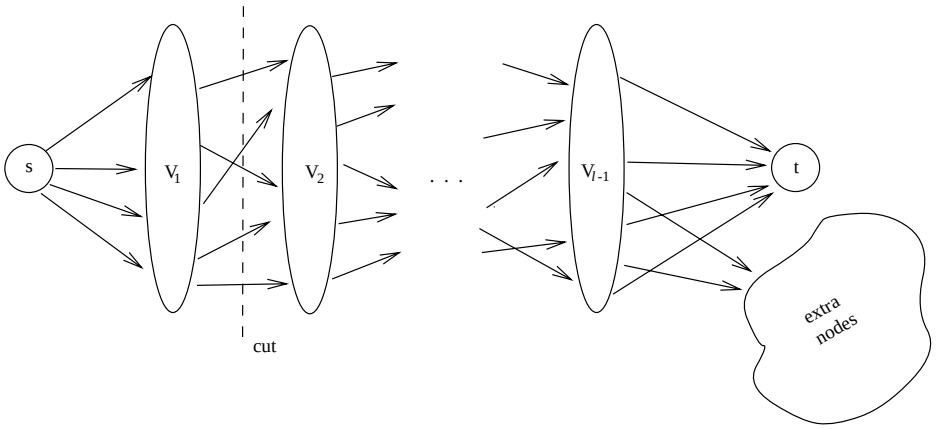

Figure 1: The layered network, distance from s to t is greater or equal to ℓ

Then $(S_i, \bar{S}_i)$ is a s-t cut. Using the fact that the maximum flow value is less than the capacity of any cut,

$$|\hat{f}| \leq |V_i||V_{i+1}| \Rightarrow \text{ either } |V_i| \geq \sqrt{|\hat{f}|} \text{ or } |V_{i+1}| \geq \sqrt{\hat{f}}$$

The first inequality comes from the fact that the maximum possible capacity of a cut is equal to $|V_i||V_{i+1}|$. Since each arc has a capacity of one, and the maximum of number of arcs from V_i toV_{i+1} is equal to $|V_i||V_{i+1}|$ (all the combinations of the set of nodes V_i and V_{i+1}).

So at least $\frac{\ell}{2}$ of the layers have at least $\sqrt{|\hat{f}|}$ nodes each
(consider the pairs, V_0V_1, $\ldots$, V_2V_3, $\ldots$).

$$\frac{\ell}{2}\sqrt{|\hat{f}|} \leq \sum_{i=1}^{\ell} |V_i| \leq |V| \Rightarrow \sqrt{|\hat{f}|} \leq \frac{2|V|}{\ell}$$

□

Claim 6. *Dinic's Algorithm, applied to the unit capacity network requires at most $O(n^{2/3})$ stages.*

Proof. Notice that in this case, it is impossible to have non-saturated arc processing, because of unit capacity, so the total computation per stage is no more than $O(m)$. If Max flow $\leq 2n^{2/3}$ then done. (Each stage increments the flow by at least one unit.)

If Max flow $> 2n^{2/3}$, run the algorithm for $2n^{2/3}$ stages. The distance labels from s to t will be increased by at least 1 for each stage. Let the Max flow in the residual network (after $2n^{2/3}$ stages) be g. The shortest path from s to t in this residual network satisfies $\ell \geq 2n^{2/3}$.

Apply lemma 5 to this residual network.

$$\sqrt{|g|} \leq \frac{2|V|}{2n^{2/3}} = n^{1/3}, |V| = n \Rightarrow |g| \leq n^{2/3}$$

It follows that no more than $n^{2/3}$ additional stages are required. Total number of stages $\leq 3n^{2/3}$. □

Complexity Analysis

Any arc processed is saturated (since this is a unit capacity network). Hence $O(m)$ work per stage. There are $O(n^{2/3})$ stages. This yields an overall complexity $O(mn^{2/3})$.

Remark: Recently (1998) A. Goldberg and S. Rao [**?**] devised a new algorithm for maximum flow in general networks that use ideas of the unit capacity as subroutine. The overall running time of their algorithm is $O(\min\{n^{2/3}, m^{1/2}\}m \log(n^2/m) \log U)$ for U the largest capacity in the network.