CS 538 Combinatorial Optimization **Fall Semester, 2020**

Maximum Flows and Min-Cost Flows. Version 1.1

1 Network Flows Definitions

A **network** is a tuple $G = (V, E, u, s, t)$, where V is a set of vertices, E is a set of directed edges (parallel edges are allowed), $s \in V$ is the **source**, $t \in V$ is the **sink**, u is a **capacity** function: $u : E \to Z_+$.

For a subset of vertices $A \subseteq V$, we denote by $\delta^-(A)$ the set of edges with head in A and tail outside A, and by $\delta^+(A)$ the set of edges with tail in A and head outside A,

Definition 1 A function $f : E \to R_+$ is called a *flow* if the following two conditions are satisfied:

1. *conservation of flow at interior vertices:* for all vertices v not in $\{s, t\}$,

$$\sum_{e \in \delta^-(v)} f(e) = \sum_{e \in \delta^+(v)} f(e) \; ;$$

2. *capacity constraints:* $f \leq u$ pointwise: i.e. for all $e \in E$,

$$f(e) \leq u(e) \; .$$

Definition 2 The value of a flow f, denoted by $|f|$, is defined to be

$$|f| = \sum_{e \in \delta^+(s)} f(e) - \sum_{e \in \delta^-(s)} f(e).$$

We say that e is **saturated** if $f(e) = u(e)$.

Definition 3 An *s-t cut* (or just *cut*, when s and t are understood) is a pair (A, B) of disjoint subsets of V whose union is V such that $s \in A$ and $t \in B$. The *capacity* of the cut (A, B), denoted by $u(A, B)$, is

$$u(A, B) = \sum_{e \in \delta^+(A)} u(e) \; .$$

If f is a flow, we define the *flow across the cut (A,B)* to be

$$f(A, B) = \sum_{e \in \delta^+(A)} f(e) - \sum_{e \in \delta^-(A)} f(e) \; .$$

Lemma 1.1 *For any $s - t$ cut (A, B), $f(A, B) = |f|$.*

Definition 4 Given a flow f on a network G define the *residual network* G_f as follows: For every $e \in E$ with $f(e) < u(e)$, add an edge e' in G_f with $u_f(e') = u(e) - f(e)$; e' is the *forward* edge obtained from e. For every $e \in E$ with $f(e) > 0$, add an edge $\overline{e}$ in G_f with $u_f(\overline{e}) = f(e)$; $\overline{e}$ is the *back* edge obtained from e. G_f has the same s and t.

Definition 5 Given a network G and flow f on G, an *augmenting path* is a directed path from s to t in the residual network G_f.

Lemma 1.2 *Given f' a flow in G_f, consider the function $\hat{f} : E \to R_+$ defined by $\hat{f}(e) = f(e) + f'(e') - f'(\overline{e})$, where e' and $\overline{e}$ are the forward and back edge obtained from e. Then $\hat{f}$ is a flow in G with $|\hat{f}| = |f| + |f'|$.*

Given $\hat{f}$ is a flow in G, consider the function $f' : E_f \to R_+$ defined as follows: if for edge $e \in E$ we have $f(e) < \hat{f}(e)$, then for the forward edge obtained from e we have $f'(e') = \hat{f}(e) - f(e)$, and if for edge $e \in E$ we have $f(e) > \hat{f}(e)$, then for the back edge obtained from e we have $f'(\overline{e}) = f(e) - \hat{f}(e)$, with f' being zero on the other edges of G_f. Then f' is a flow in G_f with $|f'| = |\hat{f}| - |f|$.

The main theorem in Network Flows is the following MaxFlow-MinCut Theorem:

Theorem 1.3 *Let $G = (V, E, u, s, t)$ be a network and f be a flow in G. The following three conditions are equivalent:*

1. *f is a maximum flow in G.*
2. *The residual network G_f contains no augmenting paths.*
3. *$|f| = u(A, B)$ for some s-t cut (A, B).*

2 The Ford-Fulkerson Algorithm

BELLMAN-FORD-FULKERSON(G,s,t)

1. **for** each edge $e \in E(G)$
2. **do** $f(e) \longleftarrow 0$
3. Construct G_f
4. **while** there exists a path P from s to t in the residual network G_f
5. **do** $u_f(P) \longleftarrow min_{e \in P} u_f(e)$
6. **for** each edge a in P
7. **do if** a is a forward edge: $a = e'$ for some $e \in E$
8. $f(e) \longleftarrow f(e) + u_f(P)$
9. **else** ($a = \overline{e}$ for some $e \in E$)
10. $f(e) \longleftarrow f(e) - u_f(P)$
11. Construct G_f

3 More properties

The "flow-decomposition theorem" is:

Theorem 3.1 *Let f be a flow in G with $|f| \geq 0$. Then there exists paths $s-t$ paths $P_1, \ldots, P_k$ with positive integers $\alpha_1, \ldots, \alpha_k$ and circuits $Q_1, \ldots, Q_r$ with positive integers $\beta_1, \ldots, \beta_r$, with $0 \leq k + r \leq |\{e \in E \mid f(e) > 0\}|$, such that for all $e \in E$,*

$$f(e) = \sum_{i \in \{1,\ldots,k\} \wedge e \in P_i} \alpha_i + \sum_{j \in \{1,\ldots,r\} \wedge e \in Q_j} \beta_j$$

and $|f| = \sum_{i=1}^{k} \alpha_i$.

The "uncrossing" of minimum s-t cuts is:

Theorem 3.2 *Let $G = (V, E, u, s, t)$ be a network, and let $(X, V \setminus X)$ and $(Y, V \setminus Y)$ be minimum $s-t$ cuts in G. Then $(X \cap Y, V \setminus (X \cap Y))$ and $(X \cup Y, V \setminus (X \cup Y))$ are also minimum $s-t$ cuts in G.*

4 Applications of the MaxFlow-MinCut Theorem: Menger's theorems

Theorem 4.1 *Let G be a graph (directed or undirected), let s and t be two vertices, and $k \in Z_+$. Then there are k edge-disjoint $s-t$ paths iff after deleting any $k-1$ edges t is still reachable from s.*

Theorem 4.2 *Let G be a graph (directed or undirected), let s and t be two non-adjacent vertices, and $k \in Z_+$. Then there are k internally-vertex-disjoint $s-t$ paths iff after deleting any $k-1$ vertices (distinct from s and t) t is still reachable from s.*

Definition 6 An undirected graph with more than k vertices is k-connected if the deletions of any $k-1$ vertices leaves a connected graph.

An undirected (multi)-graph k-edge-connected if the deletions of any $k-1$ edges leaves a connected graph.

Theorem 4.3 *An undirected graph G is k-edge-connected iff for each pair $s, t \in V(G)$ with $s \neq t$ there are k edge-disjoint $s-t$ paths.*

An undirected graph G with more than k vertices is k-connected iff for each pair $s, t \in V(G)$ with $s \neq t$ there are k internally-vertex-disjoint $s-t$ paths.

5 Minimum Cost Network Flows Version

A **network** is a tuple $G = (V, E, u, c, s, t)$, where V is a set of vertices, E is a set of directed edges (parallel edges are allowed), $s \in V$ is the **source**, $t \in V$ is the **sink**, u is a **capacity** function: $u : E \to Z_+$, and c is a **cost** function: $c : E \to R$ (note: costs can be negative).

Definition 7 The Minimum Cost Flow problem has as input a network G and a value b, and the objective of finding a flow f of value b which minimizes $\sum_{e \in E} c(e) f(e)$.

Definition 8 Given a flow f on a network G define the *residual network* G_f as follows: For every $e \in E$ with $f(e) < u(e)$, add an edge e' in G_f with $u_f(e') = u(e) - f(e)$ and $c_f(e') = c(e)$; e' is the *forward* edge obtained from e. For every $e \in E$ with $f(e) > 0$, add an edge $\overline{e}$ in G_f with $u_f(\overline{e}) = f(e)$ and $c_f(\overline{e}) = -c(e)$; $\overline{e}$ is the *back* edge obtained from e. G_f has the same s and t.

Lemma 5.1 *Given f' a flow in G_f, consider the function $\hat{f} : E \to R_+$ defined by $\hat{f}(e) = f(e) + f'(e') - f'(\overline{e})$, where e' and $\overline{e}$ are the forward and back edge obtained from e. Then $\hat{f}$ is a flow in G with $|\hat{f}| = |f| + |f'|$ and $c(\hat{f}) = c(f) + c_f(f')$.*

Given $\hat{f}$ is a flow in G, consider the function $f' : E_f \to R_+$ defined as follows: if for edge $e \in E$ we have $f(e) < \hat{f}(e)$, then for the forward edge obtained from e we have $f'(e') = \hat{f}(e) - f(e)$, and if for edge $e \in E$ we have $f(e) > \hat{f}(e)$, then for the back edge obtained from e we have $f'(\overline{e}) = f(e) - \hat{f}(e)$, with f' being zero on the other edges of G_f. Then f' is a flow in G_f with $|f'| = |\hat{f}| - |f|$ and $c_f(f') = c(\hat{f}) - c(f)$.

Theorem 5.2 *f is a minimum-cost flow of value b iff $|f| = b$ and G_f does not contain any negative-cost cycle.*

Theorem 5.3 *If f is a min-cost flow in G of value $|f|$, P is a min-cost $s - t$ path in G_f, and $\gamma \leq \min_{e \in P} u_f(e)$, then the flow $f + \gamma f_P$ is a min-cost flow in G of value $|f| + \gamma$, where f_P is the path flow in G_f shipping one unit of flow from s to t along P.*

6 Capacity Scaling Algorithm

First, assume that b (the target value of flow) is the value of maximum flow. Let U be the maximum capacity of an edge. Second, replace each edge e by at most $\log U$ edges with the same cost and capacities 2^j, for some integer j. Third, sort the edges in decreasing order of capacities: $e_1, e_2, \ldots, e_{m'}$; note $m' \leq m \log U$. Let G^i be the sub-network which only has the edges $e_1, e_2, \ldots, e_i$.

1 f is the zero flow.
2 **for** i = 1 **to** m'
3 add edge e_i (from x to y) to G_f^{i-1} to obtain G_f^i
4 Compute P, the min-cost $y - x$ path in G_f^i
5 **if** $c(P) + c(e_i) < 0$, route $u(e_i)$ units of flow on the cycle (e_i, P)
6 Compute P', the min-cost $s - t$ path in G_f^i and augment at maximum capacity on P'
7 **endfor**

The algorithm maintains the following invariants:

- if $u(e_i) = 2^j$, then all the edges in G_f when considering e_i in Step 3 have capacities multiple of 2^j.
- before executing Step 3, f is maximum flow of minimum cost in G^{i-1}.

The invariants ensures that if Step 5 does a negative-cost-cycle rerouting or Step 6 an augmentation, the amount of routed flow is exactly 2^j and therefore the first invariant is maintained. We used the fact that the augmentation cannot exceed 2^j since the capacity of the minimum cut canot increase by more than 2^j when adding e_i. This fact also shows we have a maximum flow in G_i after executing Step 6. Now:

Lemma 6.1 *After executing Step 5, f is a min-cost flow among flows of value $|f|$ in G^i.*

Thus Theorem 5.3 ensure the second invariant is maintained. Therefore the algorithm correctly computes a min-cost maximum flow; its running time is $O(m'(m'n))$, with the $O(m'n)$-time Bellman-Ford algorithm computing the min-cost paths with negative costs. This is $O(m \log U(nm \log U))$. A third invariant maintained is that all the computed flows have integer values.