Lecture Outline 5
Topics in Graph Algorithms (CSCI5320-18S)

CAI Leizhen
Department of Computer Science and Engineering
The Chinese University of Hong Kong
lcai@cse.cuhk.edu.hk

February 7, 2018

Keywords: network flow, residual graph, augmenting path, maximum matching, disjoint paths, and project selection problem.

1. **References**

   Review article by Goldberg and Tarjan, Efficient maximum flow algorithms, Communications of the ACM, 57(6), 82-89, 2014.

   Chapter 7 of Algorithm Design (Kleinberg and Tardos) contains a rich collection of applications of network flows.

   Fastest algorithm for the maximum flow problem by Orlin (2013): $O(mn)$ in general and $O(n^2/\log n)$ when $m = O(n)$.

2. **Flow network**: Weighted digraph $G = (V, E; c)$ with *capacity* $c : E \to R^+$, a *source* $s \in V$ and *sink* $t \in V$.

   We assume that every vertex in on an (s, t)-path.

3. **Notation**

   For a vertex v, $E^+(v)$ denotes the set of edges going out from v, and $E^-(v)$ the set of edges coming into v.

   An (s, t)-*cut* $[S, T]$ is a partition of V into S and T such that $s \in S$ and $t \in T$. We also use $[S, T]$ to denote edges across S and T, $[S \to T]$ edges from S to T, and $[S \leftarrow T]$ edges from T to S.

4. **Flow**: A function $f : E \to R^+ \cup \{0\}$ that satisfies the following two conditions:

   1. *capacity constraint*: for every edge e, $f(e) \le c(e)$, and
   2. *flow conservation*: for every vertex $v \in V - \{s, t\}$,

   $$\sum_{e \in E^-(v)} f(e) = \sum_{e \in E^+(v)} f(e).$$

   The *value* $|f|$ of flow f is defined as $\sum_{e \in E^+(s)} f(e)$.

5. **Residual graph**: For a flow f of G, the residual graph $G_f = (V, E_f; c_f)$ is defined as follows: for each edge $e = uv$ of G, if $f(e) > 0$ then vu is an edge in G_f with *residual capacity* $c_f(vu) = f(e)$, and if $c(e) - f(e) > 0$ then uv is an edge of G_f with *residual capacity* $c_f(uv) = c(e) - f(e)$. Note that $|E_f| \le 2|E|$.

6. **Augmenting path**: a simple (s,t)-path P in G_f. The *residual capacity* of P: $c_f(P) = \min\{c_f(uv) : uv \in P\}$.

   We can use an augmenting path P to define a new flow f' with $|f'| > |f|$:

   For each edge e of G, set $f'(e) = f(e) + c_f(P)$ if e is in P, $f'(e) = f(e) - c_f(P)$ if the reverse of e is in P, and $f'(e) = f(e)$ otherwise.

7. Let $[S,T]$ be an (s,t)-cut.

   Net flow across $[S,T]$: $f(S,T) = \sum_{[S\to T]} f(e) - \sum_{[S\leftarrow T]} f(e)$.

   Capacity of $[S,T]$: $c(S,T) = \sum_{[S\to T]} c(e)$.

   Lemma. For any flow f and any (s,t)-cut $[S,T]$, $f(S,T) = |f| \le c(S,T)$.

8. **Theorem** (Min-Cut Max-Flow) The following statements are equivalent:

   (a) f is a maximum flow.

   (b) G_f has no augmenting path.

   (c) $|f| = c(S,T)$ for some (s,t)-cut $[S,T]$.

   Proof. (a) $\to$ (b). If G_f has an augmenting path P, then we has a new flow f' with $|f'| > |f|$.

   (b) $\to$ (c). Let $S = \{v \in V : \text{there is an } (s,v)\text{-path in } G_f\}$ and $T = V - S$. Since G_f has no (s,t)-path, $[S,T]$ forms an (s,t)-cut and no flow goes from T to S. Furthermore, for every edge e going from S to T, we have $f(e) = c(e)$. Therefore $|f| = f(S,T) = c(S,T)$.

   (c) $\to$ (a). Since $|f| \le c(S,T)$, f is indeed a maximum flow.

9. **Ford-Fulkerson Algorithm** (1962)

   $f \leftarrow 0$;

   while there is an augmenting path P in G_f **do** augment f to f' using P.

   For integer-valued capacities, the algorithm always returns *a maximum flow with every flow value being an integer*, and runs in $O(m|f^*|)$ time, where f^* is a maximum flow. However the algorithm may not terminate for real-valued capacities.

   Edmonds-Karp algorithm ($O(m^2 n)$ time): an implementation of Ford-Fulkerson algorithm by using a shortest (s,t)-path (in terms of the number of edges in the path) in G_f as an augmenting path. This algorithms works for real-valued capacities.

   Remark. If all capacities are integers, then Ford-Fulkerson and Edmonds-Karp algorithms always find a maximum flow whose value is an integer.

10. **Applications**: The maximum flow (minimum cut) algorithm is very useful for solving a large number of problems.

11. **Maximum matching** in bipartite graphs.

    A *matching* is a subset of edges that are mutually nonadjacent. To find a maximum matching in a bipartite graph $G = (X, Y, E)$, we construct a flow network G' from G as follows: Add source s and sink t, add edge sx for every $x \in X$ and edge yt for every $y \in Y$, and orient every edge of G from X to Y. Set the capacity of every edge to 1.

    The value of a maximum flow of G' equals the size of a maximum matching of G.

12. **Disjoint paths**: Find the maximum number of edge-disjoint (s,t)-paths in digraph G.

    Set the capacity of every edge to 1, and use Ford-Fulkerson algorithm.

    We can also use the following construction to find the maximum number of vertex-disjoint (s,t)-paths: For each vertex v, split it into two vertices v_{in} and v_{out}, change all edges coming into v to coming into v_{in} and all edges going out of v to going out of v_{out}, and add edge $v_{in}v_{out}$. Set the capacity of every edge to 1.

13. **Project selection**: Given a dag (directed acyclic graph) $G = (V, E; w)$ with $w : V \to Z - \{0\}$, we want to find a subset V' of vertices to maximize $\sum_{v \in V'} w(v)$ subject to *precedence constraint*: for each $v \in V'$, every out-neighbor of v is also in V'.

    Background: Each vertex v represents a project, and weight $w(v)$ is the profit for pursuing project v, and an edge uv indicates that in order to pursue project u we must also pursue project v. We wish to select a feasible set of projects with maximum profit.

    The difficulty of the problem is caused by projects with negative profits.

    Construct a flow network $G' = (V', E'; c')$ as follows:

    - Add a source s and a sink t.
    - For each vertex u with $w(u) > 0$, add edge su with capacity $w(u)$.
    - For each vertex v with $w(v) < 0$, add edge vt with capacity $-w(v)$.
    - Set the capacity of each edge of G to ∞.

    We compute a minimum (s,t)-cut $[S, T]$ of G', and output $S - \{s\}$ as an optimal set of projects.

    Let C be the capacity of the cut $[\{s\}, V' - \{s\}]$ in G', i.e., $C = \sum_{w(v)>0} w(v)$. Then the maximum flow of G' has value at most C.

    Lemma. Let $[S, T]$ be an (s,t)-cut of G'.

    (a) If $c(S,T) \le C$ then $S - \{s\}$ satisfies the precedence constraint.

    (b) If $S - \{s\}$ satisfies the precedence constraint, then the capacity of $[S, T]$ equals $C - \sum_{v \in S - \{s\}} w(v)$.

    The above lemma implies that when $[S, T]$ is a minimum (s,t)-cut of G', $S - \{s\}$ yields an optimal solution.

Proof of the lemma. Let S^* be a subset of vertices satisfying the precedence constraint. Set $S = S^* \cup \{s\}$ and $T = V' - S$. Then in G', $[S \to T]$ consists of edges leaving s or entering t only. For capacity $c(S,T)$, edges entering t contribute

$$\sum_{v \in S^* \text{ and } w(v)<0} -w(v),$$

and edges leaving s contribute

$$\sum_{v \notin S^* \text{ and } w(v)>0} w(v) = C - \sum_{v \in S^* \text{ and } w(v)>0} w(v),$$

where $C = \sum_{w(v)>0} w(v)$. Therefore (b) holds, i.e.,

$$c(S,T) = C - \sum_{v \in S^*} w(v).$$

Since every edge of G has capacity ∞, $c(S,T) \leq C$ implies that no edge of G goes from S to T. Therefore $S - \{s\}$ satisfies the precedence constraint, and (a) holds. ∎