Lecture notes for Feb 27, 2023
Decomposition of flows and Bipartite matching

Chun-Hung Liu

February 27, 2023

1 Decomposing a flow into paths

Flows can be view as a conic combination of directed paths:

Proposition 1 *Let (D, s, t, c) be a network. Let f be a feasible flow in (D, s, t, c). Then there exist a multiset $\mathcal{F}$ of directed paths in D from s to t, and a function $g : \mathcal{F} \to \mathbb{R}_{\geq 0}$ such that*

1. *$\sum_{P \in \mathcal{F}} g(P) = \mathrm{val}(f)$, and*
2. *for every $e \in E(D)$, $f(e) = \sum_{P \in \mathcal{F}, e \in E(P)} g(P)$.*

Moreover, if f is integral, then $|\mathcal{F}| \leq \mathrm{val}(f)$ and the image of g is a subset of $\mathbb{N}$. Furthermore, $\mathcal{F}$ and g can be found in time $O(|\mathcal{F}| \cdot (|V(D)| + |E(D)|))$.

Proof. Let D_f be the digraph with $V(D_f) = V(D)$ and $E(D_f) = \{e \in E(D) : f(e) > 0\}$. We do induction on $|E(D_f)|$. If $E(D_f) = \emptyset$, then $\mathrm{val}(f) = 0$, so we are done by choosing $\mathcal{F} = \emptyset$. So we may assume $E(D_f) \neq \emptyset$ and $\mathrm{val}(f) > 0$.

Let $A = \{v \in V(D) : \text{there exists a directed path in } D_f \text{ from } s \text{ to } v\}$. If $t \notin A$, then $\mathrm{val}(f) = \sum_{e \in \delta^+(A)} f(e) - \sum_{e \in \delta^-(A)} f(e) = 0 - \sum_{e \in \delta^-(A)} f(e) \leq 0$, a contradiction. So $t \in A$. Hence there exists a directed path P in D_f from s to t. Let e_P be an edge of P with $f(e_P) = \min_{e \in E(P)} f(e)$. Let f' be the function such that $f'(e) = f(e) - f(e_P)$ for every $e \in E(P)$, and $f'(e) = f(e)$ for every $e \in E(D) - E(P)$. Then f' is clearly a flow in (D, s, t, c) with $\mathrm{val}(f') = \mathrm{val}(f) - f(e_P)$, and $E(D_{f'}) \subseteq E(D_f) - \{e_P\}$. By the induction

hypothesis, we obtain a set $\mathcal{F}'$ of paths and a function g' for the flow f' as desired. Let $\mathcal{F} = \mathcal{F}' \cup \{P\}$, and let $g(P) = f(e_P)$ and $g(P')$ for every $P' \in \mathcal{F}'$. Then $\mathcal{F}$ and g are desired for the flow f.

Moreover, if f is integral, then $\mathrm{val}(f)$ and $f(e_P)$ are integral, so f' is also integral and the image of g' is contained in $\mathbb{N}$, and hence $|\mathcal{F}| = |\mathcal{F}'| + 1 \leq \mathrm{val}(f') + 1 \leq \mathrm{val}(f') + f(e_P) = \mathrm{val}(f)$ and the image of g is contained in $\mathbb{N}$.

Note that finding a path P mentioned above takes time $O(|V(D_f)| + |E(D_f)|) = O(|V(D)| + E(D)|)$. And we only needs to find $|\mathcal{F}|$ those paths. So we can find $\mathcal{F}$ and g in time $O(|\mathcal{F}| \cdot (|V(D)| + |E(D)|))$. ∎

2 Matching

A *matching* in a graph G is a subset M of $E(G)$ such that no two edges in M share an end. A matching M is *maximal* if there exists no matching M' with $M' \supset M$. A matching M is *maximum* if there exists no matching M' with $|M'| > |M|$. Clearly, every maximum matching is maximal, but some maximal matching is not maximum. We can find a maximal matching by an easy greedy algorithm in linear time. So finding a maximal matching is not interesting. We will consider how to find a maximum matching.

2.1 Maximum matching in bipartite graphs

Finding a maximum matching in a bipartite graph is actually a special case of the maximum flow problem.

Let's recall an easy observation.

Proposition 2 *Let G be a graph. Let M be a matching in G. Let S be a vertex-cover of G. Then $|S| \geq |M|$. In particular, the maximum size of a matching in G is at most the minimum size of a vertex-cover in G.*

Proof. Every edge in M is incident with at least one vertex in S. Since edges in M do not share ends, no two edges in M are incident with the same vertex in S. So $|S| \geq |M|$. ∎

Corollary 3 *Given a bipartite graph G, we can find a matching M in G and a vertex-cover S of G with $|M| = |S|$ in time $\theta(|V(G)|+2, |E(G)|+|V(G)|)$, where $\theta(x, y)$ is the running time for finding a maximum flow in a network with digraph on x vertices with y edges. In particular, M is a maximum*

matching and S is a minimum vertex-cover, and they can be found in time $O(|V(G)||E(G)|^2)$.

Proof. Using BFS, we can find a bipartition $\{A, B\}$ of G in linear time. Let D be the digraph with $V(D) = V(G) \cup \{s, t\}$ and $E(D) = \{(a, b) : ab \in E(G)\} \cup \{(s, a) : a \in A\} \cup \{(b, t) : b \in B\}$. That is, we add two new vertices s, t to G, where s points to all vertices in A, and t is pointed by all vertices in B, and direct edges of G from A to B. Note that D can be constructed in linear time.

Let $c(e) = 1$ for every $e \in E(D)$. Use Edmond-Karp's algorithm (or any other applicable algorithm) to find a maximum flow and a minimum cut in (D, s, t, c) in time $O(|V(D)||E(D)|^2) = O(|V(G)||E(G)|^2)$. Since c is integral, it outputs an integral maximum flow f in (D, s, t, c). By Proposition 1, there exist a multiset $\mathcal{F}$ of directed s-t paths in D, and a function $g : \mathcal{F} \to \mathbb{N}$ such that $\sum_{P \in \mathcal{F}} g(P) = \text{val}(f)$ and $f(e) = \sum_{P \in \mathcal{F}, e \in E(P)} g(P)$. Since c is integral, $f(e) \in \{0, 1\}$ for every $e \in E(D)$. So the image of g is $\{1\}$ and $\mathcal{F}$ is a set (instead of a multiset). Moreover, since $c = 1$ is a constant function, $|\mathcal{F}| = \text{val}(f) \leq |A| \leq |V(G)|$, so the proof of Proposition 1 gives a $O(|V(G)| \cdot (|V(G)| + |E(G))))$ time algorithm for finding $\mathcal{F}$.

Since $g(e) = 1$ for every $e \in E(D)$, every vertex in $A \cup B$ is contained in at most one path in $\mathcal{F}$. Hence paths in $\mathcal{F}$ are internally disjoint. Note that every path in $\mathcal{F}$ contains at least one edge between A and B. So by deleting s and t, the paths in $\mathcal{F}$ gives a matching M^* in G with size $|\mathcal{F}| = \text{val}(f)$.

Let C be the minimum cut in (D, s, t, c) output from the algorithm. That is, $C \subseteq V(D)$ with $s \in C$ and $t \notin C$, and the number of edges of D from C to $V(D) - C$ equals $\text{val}(f) = |\mathcal{F}| = |M^*|$. Let E_1 be the set of edges of D from s to $A - C$. Let E_2 be the set of edges of D from $A \cap C$ to $B - C$. Let E_3 be the set of edges of D from $B \cap C$ to t. Note that $\{E_1, E_2, E_3\}$ is a partition of the set of edges of D from C to $V(D) - C$. So $|E_1| + |E_2| + |E_3| = |M^*|$.

Note that $|E_1| = |A - C|$ and $|E_3| = |B \cap C|$. Let S' be the set of the ends in A of the edges from $A \cap C$ to $B - C$. So $|S'| \leq |E_2|$. Let $S = (A - C) \cup S' \cup (B \cap C)$. Then $|S| = |A - C| + |S'| + |B \cap C| \leq |E_1| + |E_2| + |E_3| \leq |M^*|$.

Note that every edge of G incident with $A - C$ or $B \cap C$ is incident with S. And every edge between $A \cap C$ and $B - C$ is incident with $S' \subseteq S$. So S is a vertex-cover of G.

By Proposition 2, $|S| \geq |M^*|$. Since $|S| \leq |M^*|$, $|S| = |M^*|$. ∎

The time complexity in Corollary 3 can be improved, and we will see it later. Corollary 3 also gives the following immediate corollaries.

Corollary 4 (König) *Let G be a bipartite graph. Then $\max_M |M| = \min_S |S|$, where the maximum is over all matchings M in G and the minimum is over all vertex-cover S in G.*

Corollary 5 *Given a bipartite graph G, a vertex-cover of G with minimum size can be found in polynomial time.*

Recall that finding a minimum vertex-cover of a general graph is NP-hard. But it becomes polynomial if we restrict the input graph to be bipartite, as shown in Corollary 5.

Let's review our proof via the algorithm for maximum flows. During the algorithm, we have an integral flow in the network (D, s, t, c), which is equivalent to having a matching in G, and we try to find an integral flow with larger value (equivalently, finding a matching with larger size) by finding a flow-augmenting path. Since $c(e) = 1$ for every $e \in E(D)$, for each flow-augmenting path, after deleting s and t, it uses edges of D with value-zero and edges with value-one alternatedly, and the first edge and the last edge must have value-zero. That is, each flow-augmenting path (after deleting s and t) uses edges of G such that edges in M and edges in $G - M$ appear alternatedly, and the first vertex and the last vertex are not saturated by M. And we obtain a maximum flow (equivalently, a maximum matching) if we cannot find such a flow-augmenting path (equivalently, a path in G described above). It leads to the following notions.

Let G be a graph. Let M be a matching. A path P in G is an *M-alternating path* if P uses edges in M and $G - M$ alternatedly. A path P in G is an *M-augmenting path* if it is an M-alternating path, and the ends of P are not saturated by M.

Let Δ be the operator that denotes the symmetric difference. That is, if A and B are sets, then $A \Delta B$ is the set $(A - B) \cup (B - A)$. Note that if P is an M-augmenting path in a graph G for some matching M, then $M \Delta E(P)$ is a matching with size $|M| + 1$.

Proposition 6 *Let G be a graph (not necessarily bipartite). Let M be a matching. Then M is a maximum matching if and only if there exists no M-augmenting path in G.*

Proof. ($\Rightarrow$) If there exists an M-augmenting path P, then $M \Delta E(P)$ is a matching with size $|M| + 1$, so M cannot be a maximum matching.

($\Leftarrow$) Let M^* be a maximum matching. Let H be the graph with $V(H) = V(G)$ and $E(H) = M \Delta M^*$. So every vertex of H has degree at most 2. Hence H is a disjoint union of paths and cycles. It is easy to see that for every component Q of H, $|E(Q \cap M^*)| > |E(Q \cap M)|$ if and only if Q is an M-augmenting path. Since we assume that there is no M-augmenting path, $|E(Q \cap M^*)| \leq |E(Q \cap M)|$ for every component Q of H. Hence $|M^*| = \sum_Q |E(Q \cap M^*)| \leq \sum_Q |E(Q \cap M)| = |M|$, where both sums are over all components Q of H. Therefore, M is also a maximum matching. ∎

Hence to find a maximum matching, we can start with an arbitrary matching M, and then use BFS to either find an M-augmenting path and get a matching of larger size by taking symmetric difference, or conclude that there is no M-augmenting path and M is maximum.

Formally, we start with a matching M and try to find M-augmenting path by using a variant of BFS. We construct a tree T rooted at an unsaturated vertex r. We see all the neighbors of r and adding the edges incident with r into the tree T. Since r is unsaturated, all those edges are not in M. If some neighbor of r is unsaturated, then we get an augmenting path, so we can enlarge the matching. So we may assume every neighbor of r is saturated. Then we add the edges in M incident with those vertices into T. Now the newly added vertices of T have all other incident edges not in M, so they behave like r in the sense that all neighbors not in T are adjacent to them by using edges not in M, and then we add them into the tree T and repeat until we cannot new vertices into T. Then we pick a unsaturated vertex not contained in T, and construct another tree rooted at it as we did before. Repeat this until we have grow the forest for all unsaturated vertices. The vertices in the forest behaved like r are called *outer vertices* (i.e. they are able to add new vertices by using non-M-edges), and other vertices in the forest are called *inner vertices* (i.e. they add other vertices by using edges in M). It can be shown that if we do not find an M-augmenting path during this process, then there is indeed no M-augmenting path, so M is maximum. The analysis of this algorithm is simple because of the bipartiteness.

This algorithm does not explicitly use flows, but it is essentially the same as the one by using flows. And it can be shown that this algorithm runs in time $O(|V(G)||E(G)|)$.