CSC373S Lecture 17

- We recall the definition of a flow in a flow network.

  A flow in $\mathcal{F}$ is a function $f : E \to \Re$ (not necessarily non negative) satisfying the following conditions:

  1. capacity constraint $f(e) \leq c(e)$ for all edges e.
  2. skew symmetric $f(u,v) = -f(v,u)$ IN KT, $f(u,v) \geq 0$
  3. flow conservation for all $u \neq s,t$ $\sum_{v:(u,v)\in E} f(u,v) = 0$. In KT, sum in = sum out

  A little thought shows that this is indeed an equivalent formulation of the more direct "flow in = flow out" statement of flow conservation.

  The value of a flow, which I will denote by $val(f)$ or $|f|$ is $val(f) = \sum_{v\in N(s)} f(s,v)$ = flow out of s.

  Later we will see that this is the same as flow into t; that is, $\sum_{u:(u,t)\in E} f(u,t)$

- Consider the example in the old lecture notes. (Convention: negative flows on 0 capacity edges are not shown.)

  Let f be a flow in flow network $\mathcal{F} = (G, c, s, t)$

  The *residual capacity* (wrt. flow f) on an edge e is $c_f(e) = c(e) - f(e) \geq 0$ (by the capacity constraint). The *residual graph* $G_f = (V, E_f)$ where $E_f = \{e | c_f(e) > 0\}$ and residual network $\mathcal{F}_f = (G_f, c_f, s, t)$. An augmenting path π (relative to a flow f) is an s to t path in G_f. An augmenting path π has a residual capacity $c_f(\pi) = min_{e\in\pi} c_f(e) > 0$.

  An augmenting path defines a flow:

  $$\begin{aligned} f_\pi &= c_f(e) \text{ if } e = (u,v) \in \pi \\ &= -c_f(e) \text{ if } e = (v,u) \text{ and } (u,v) \in \pi \\ &= 0 \text{ otherwise} \end{aligned}$$

  f_π is a flow in $\mathcal{F}_f$ (and hence in $\mathcal{F}$)

  Fact: A flow f' in $\mathcal{F}_f$. Furthermore, $(f + f')$ is a flow in $\mathcal{F}$ where $(f + f')(e) = f(e) + f'(e)$.

- The fact that an augmenting path in the residual graph allows us to increase the flow suggests the Ford Fulkerson *algorithm template*:

  For all $e \in E$, set $f(e) = 0$ % Initialize flow f
  While there exits an augmenting path in G_f
  choose an augmenting path π

$f := f + f_\pi$
End While

NOTE: I have called this an algorithm template since we have not said how such a path is chosen and if the choice is important. In particular, we may want to choose the augmenting path π so as to maximize $c_f(\pi)$. As we have seen (in the first assignment) , this can be done using Dijkstra's algorithm say in time $O(|E| \log n)$. Another possibility is choosing a shortest length path.

Two big questions:

1) Does the algorithm always terminate?
2) If it terminates, does it always yield an optimal flow?

Note: If there is an augmenting path π then clearly the flow is not optimal since the flow $c_f(\pi)$ on this path can be added to the current flow. But if there is no augmenting path does this mean that the flow is optimal?

- The max flow-min cut theorem

  An (s,t) cut is a partition $(S, V-S)$ such that $s \in S$ and $t \in T = V - S$.

  The capacity of a cut is $c(S,T) = \sum_{(x,y)\in E: x\in S, y\in T} c(x,y)$
  The flow across a cut is $f(S,T) = \sum_{(x,y)\in E: x\in S, y\in T} f(x,y)$

  Clearly $f(S,T) \leq c(S,T)$

  Claim: Let f be a flow. Then for every cut $(S,T), val(f) = f(S,T)$.

  (Lemma 3 in notes)

  Cor: $val(f) = \sum_{u,t} f(u,t) =$ flow into terminal t.

  Cor: For every cut $(S,T), val(f) \leq c(S,T)$.

  Now we are ready for the max flow-min cut theorem (Ford Fulkerson)

  Theorem: The following are equivalent:

  1) f is a max flow
  2) There are no augmenting paths wrt f
  3) $val(f) = c(S,T)$ for some cut (S,T).
  (That is, we can conclude "max flow = min cut").

  Proof: Show 1) implies 2) implies 3) implies 1)

  1) implies 2) This is obvious as we have already mentioned that the existence of an augmenting path implies the flow is not optimal.

2) implies 3) Let $S = \{u|$ there is a path from s to u in $G_f\}$. This is a cut since $t \notin S$. $val(f) = val(S,T)$ and we claim $val(S,T) = c(S,T)$ or else some other vertex can be reached.

3) implies 1) Suppose $val(f) = c(S,T)$. Since for every cut (S',T') we have $val(f) \leq c(S',T')$, and hence (S,T) must be a max flow.

- There are some immediate corollaries of the Ford Fulkerson (FF) Min Cut Max Flow theorem and any FF algorithm. .

  1. If an FF algorithm terminates, it terminates in an optimal flow.
  2. If all the capacities are integral, then a FF alg must terminate and will compute an integral max flow. If the largest possible flow is C then any FF algorithm will terminate in time C iterations and time $O(mC)$. Note: $C \leq \sum_{e=(s,u)\in E} c_e$ and $C \leq \sum_{e=(v,t)\in E} c_e$.

- We now know that in a flow network with integral capacities, any Ford Fulkerson (FF) algorithm must terminate and terminate with an optimal flow value. This leaves the important questions as to whether or not the way we choose augmenting paths is important. We have the following facts about bad ways to choose augmenting paths:

  1. There is a way to choose augmenting paths so that even with integral capacities, it can take $val(f)$ iterations to terminate and this value can be exponential in the representation of the input (i.e. when the edge capacities are say $m = |E|$ bits meaning the flow values can be exponential in m and the input can be represented in m^2 bits. See notes for such a bad example.
  2. What is less obvious is that for some setting of algebraic (not rational) edge capacities, here is a bad way to choose augmenting paths so that FF algorithm does not terminate.

- However, the good news is that there are good ways to choose the augmenting paths so that the FF algorithm will always terminate in $poly(m)$ steps where $m = |E|$. This is called a strongly polynomial algorithm when the time does not depend on the length of the integer (or rational) inputs (except that we are counting each arithmetic operation as one step). The text gives one strongly polynomial algorithm (preflow push (with time bound $O(n^3)$ and one non strongly polynomial time algorithm (scaling max-flow with time bound $O(m^2 \log C)$ for max flow.

  It is also the case that choosing the augmenting path that maximizes the residual flow (ie using a variant of Dijkstra's algorithm that we saw in the first problem set) will lead to a strongly polynomial algorithm. In that variant we defined the cost of a path π to be the residual capacity $c_f(\pi) = min_{e\in\pi} c_f(e) > 0$.

  And choosing the augmenting path to be the shortest path also results in a strongly poly time algorithm for max flow. This is called the Edmonds-Karp algorithm and the algorithm has time complexity $O(m^2 n)$. Independently a similar result was

given by Dinitz. The Dinitz algorithm and proof is in terms of building a layered graph from the residual graph and the concept of blocking flows and is perhaps the conceptually simplest proof and has time $O(mn^2)$. When all capacities are unit, the algorithm runs in time $O(m\sqrt{n})$ time.

The current best max flow alg runs in time $O(mn)$. The max flow problem continues to be of active interest.

- Some immediate applications of max flow.
  We do not know of an efficient way to use greedy, divide and conquer (DC), or dynamic programming (DP) to optimally solve max flows nor do we know how to optimally compute maximum matchings in a bipartite graph efficiently by the previous paradigms (greedy, DC, DP).

  We are given $G = (V, E)$ a bipartite graph where $V = X \cup Y$ and $E \subseteq X \times Y$. (As defined before) a matching (in any graph) is a subset of edges $M \subseteq E$ such that no vertex is adjacent to more than one edge in M. In a bipartite graph, a matching becomes a 1-1 mapping from X into Y. The goal is to compute a matching M of maximum size $|M$—. In weighted max matching the edges have weights and the goal is to compute a matching of max weight.

  We transform the bipartite max matching problem into a flow problem as follows:

  Let $G' = (V', E')$ where $V' = V \cup \{s, t\}$ and $E' = E \cup \{(s, x)|x \in X\} \cup \{(y, t)|y \in Y\}$

  We make this into a flow network $\mathcal{F}_G$ by making s and t the source and terminal nodes and setting $c(e) = 1$ for all $e \in E'$.

  Claim: Let f be a max flow in $\mathcal{F}_G$ and M be a max matching in G, then $val(\mathcal{F}_G) = |M|$. More generally, every matching M in G gives rise to an integral flow f_M in $\mathcal{F}_G$ with $val(f_M) = |M|$ and conversely every flow f in $\neg F_G$ gives rise to a matching M_f in G with $|M| = val(f)$.

  Since $C = min|X|, |Y|$, the algorithm terminates in $O(mC) = O(mn)$ time. Dinitz's levelled graph/blocking flow algorithm for max flow was utilized by Hopcroft and Karp to compute bipartite max matching in $O(m\sqrt{n})$ which remains as the best known time bound for bipartite max matching.

  Corollaries: Hall's Theorem as corollary of max flow-min cut: In a bipartite graph $G = (V, E)$ with vertices $X \cup Y$, there is a matching covering all vertices in X (and hence a perfect matching if $|X| = |Y|$) iff for all subsets $A \subseteq X$, $|N(A)| \geq |A|$.