CS 6200: Algorithmics II	**Fall 2020**
Instructor: Avah Banerjee, S&T.	
Lecture: 7	**Date: 9/9**

1 Proving Optimality

Picking up where we left off in the last lecture, we had just discussed a greedy algorithm that looks at finding the maximum weight basis (cycle matroid), which is equivalent to solving the minimum spanning forest problem. In general, if one wants to prove the optimally of a greedy algorithm, there are two main components to consider:

1. Optimal substructure
2. Greedy Choice Property: There must exist a optimal solution that is consistent with the greedy choice (that is made in the first step of the algorithm)

For example, if an edge (uv) is in the MSF of G, and T is a MSF of $G|\{uv\}$, then the MSF of G is a subset of $T + \{uv\}$.

1.1 Exchange Argument

The exchange argument is another method for proving that a greedy algorithm is optimal. Our claim is that an greedy algorithm computes an optimal solution. This exchange argument uses matroids as the device to prove our claim. The goal is to find the maximum weight independent set, where each matroid ($e \in E$) has a weight $w(e) > 0$. We start with two algorithms, a Greedy one and an optimal one:

1. GREEDY $= (g_1, g_2, ..., g_n)$ (sequence of actions)
2. OPT $= (o_1, o_2, ..., o_n)$

$$\forall\, i, solution(g_1, g_2, ..., g_k) \geq solution(o_1, o_2, ..., o_k) \tag{1}$$

In the context of matroids, let $A_k = \{a_1, ..., a_k\}$ be the set of elements picked by the greedy algorithm before iteration $k + 1$. We also have the following relationship in terms of weights:

$$[w(a_1) \geq w(a_2) \geq ... \geq w(a_k)] \tag{2}$$

We can assume this equation (3) without loss of generality. This equation combined with what we know already means that $A_k \in \mathcal{I} \,\forall\, k$. Suppose $O_n = \{O_1, ..., O_n\}$ to be an optimal $solution^n$ where $n =$ size of a basis. This results in similar weight equivalence to what we have in (2):

$$[w(o_1) \geq w(o_2) \geq ... \geq w(o_n)] \tag{3}$$

We want to show that $\forall\, k \in \{1, n\}, w(a_k) \geq w(o_k)$.

Proof: Base case: $k = 0$. This is trivially true. Suppose this holds up to $k < n$. We will use contradiction for our proof by assuming that this base case does not hold for $k+1$. So now we have $w(a_{k+1}) \leq w(o_{k+1})$. Continuing, we define the current solution as A_k (which is greedy of course), and the next solution as O_{k+1} (which is optimal), meaning that $|A_k| < |O_{k+1}|$. We can also say that these two sets (A_k and O_{k+1}) are both independent. Now using the augmentation property, we can say:

$$\exists\, o_i \in O_{k+1} |\ A_k \ s.t.\ A_{k+1} \cup \{o_i\} \in \mathcal{I} \tag{4}$$

Building on equation 4 above, we know that $i \leq k+1$, so that means that $w(o_i) \geq w(o_{k+1}) > w(a_{k+1})$. This means the our greedy algorithm looks at o_k before a_k, therefore greedy will add o_k. Thus we have a contradiction. □

2 Minimum Cut

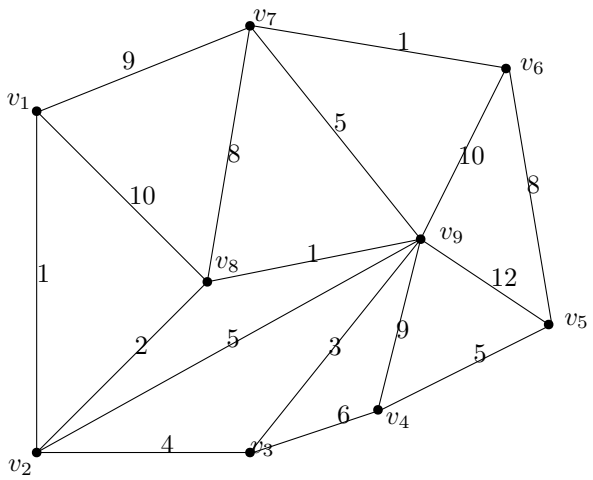

Fig. 1: Example of a Weighted Graph

Pictured above is an example of a graph $G(V, E)$, which has a weight function $w : E \Rightarrow \mathbb{R}^+$ on its edges. A cut of G is defined as any bipartition (S, T) of the vertices (S, T are both non-empty). We can also define the set of edges crossing the cut as $E(S, T)$. The weight of the cut is defined as $w(S, T)$, meaning $w(S, T) = \sum_{e \in E(S,T)} w(e)$. For example, using the graph in figure 1 if $S = \{v_7, v_8\}$ and $T = V - S$, the weight of the cut is equal to the sum of all of the weights of the edges on the cut (connected to the S vertices). Hence in this example $w(S, T) = 28$.

A minimum cut is characterized by a cut of G that has the minimum possible total weight. The min-cut problem is as following: Given $G(V, E, w)$, find a min-cut of G.

2.1 Definition of a min-cut

A $\min(s, t)$-cut is a cut of minimum weight where s and t are in different parts of the partition. Using figure 2 below, let's suppose the $s = v_2$ and $t = v_9$. Thus, $S = \{s = v_2\}$ and $T = \{v_1, v_3, ..., v_9\}$, and we can calculate the weight of that cut as $w(s, T) = 1 + 2 + 5 + 4 = 12$.

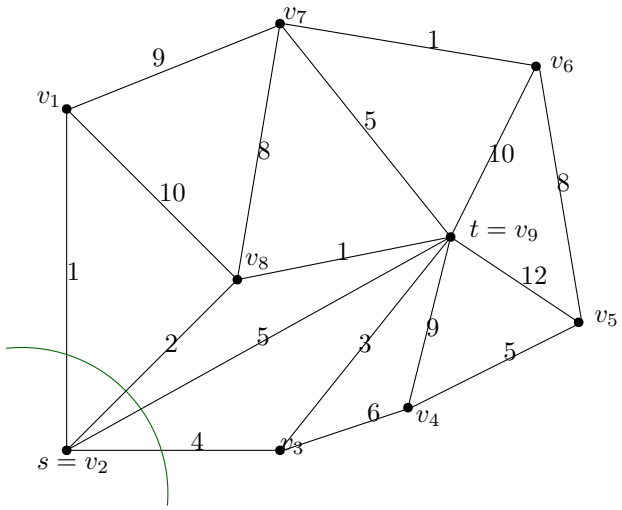

Fig. 2: Weighted Graph with Cut

We can also calculate that cut by finding the maximum flow between s and t (more on this in the coming weeks). If we use a maximum flow subroutine then the runtime will be $O(n^2 MAXFLOW) = O(n^3 m)$ (the fastest $MAXFLOW$ algorithm takes $O(nm)$ time). These algorithms are pretty complicated as seen by their runtimes.

Remark 2.1 *Can we avoid using flow algorithms?*

(hint, hint...the answer is yes)

2.2 Stoer-Wagner Algorithm

Below we see an example of $\{s, t\}$-contraction, where the vertices s and t are combined into one vertex. If the graph on the left is defined as G, then our contracted graph on the right is defined as $G/\{s, t\}$.

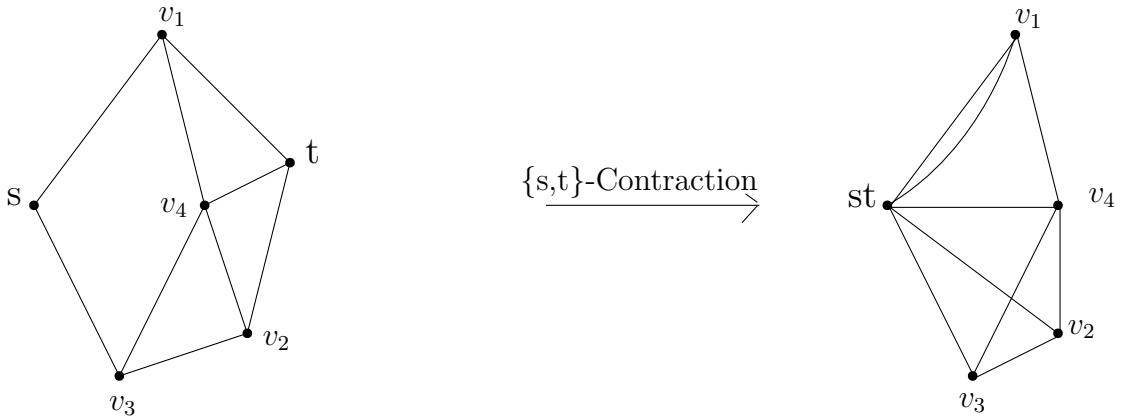

Fig. 3: Example of as $\{s, t\}$-contraction

The Stoer-Wagner Algorithm uses this idea of a contraction to formulate its main theorem, which is stated below.

Theorem 2.1 *Let s and t be two vertices of a graph G. Let $G/\{s,t\}$ be the graph obtained by merging s and t. Then a minimum cut of G can be obtained by taking the smaller of a minimum (s,t)-cut of G and a minimum cut of $G/\{s,t\}$.*

The above theorem leads to the following iterative (proto) algorithm:

Algorithm 1 Iterative Cut Algorithm (MaxFlow)

```
1: Let C* be the best cut found so far
2: while |G| ≥ 2 do
3:    find a cut in G which is the minimum for some pair, say (s, t)
4:    if w(c) < w(c*) then
5:       replace c* with c
6:    end if
7:    contract (s, t) and let G ← G/{s, t}
8: end while
```