CS5275 – The Algorithm Designer’s Toolkit (S2 AY2025/26)

Lecture 9:

Expanders – vertex connectivity

Vertex connectivity

A **vertex cut** is a partition of the vertex set V into three parts:

$$V = L \cup S \cup R$$

such that $E(L, R) = \emptyset$.

The **size** of a vertex cut (L, S, R) is $|S|$.

$L \neq \emptyset, \qquad R \neq \emptyset$

Vertex connectivity

A **vertex cut** is a partition of the vertex set V into three parts:

$$V = L \cup S \cup R$$

such that $E(L, R) = \emptyset$.

$L \neq \emptyset, \quad R \neq \emptyset$

The **size** of a vertex cut (L, S, R) is $|S|$.

The **vertex connectivity** $\kappa(G)$ of a graph is the minimum size of a vertex cut.

$n - 1$ if G is a clique.

Vertex connectivity

A set $S \subseteq V \setminus \{s,t\}$ is an $(\boldsymbol{s},\boldsymbol{t})$**-vertex cut** if $\underbrace{\text{the removal of } S \text{ disconnects } s \text{ and } t}_{}$.

Equivalently, there is a vertex cut (L, S, R) with $s \in L$ and $t \in R$

The $(\boldsymbol{s},\boldsymbol{t})$**-vertex connectivity** $\kappa_G(s,t)$ is the minimum size of an (s,t)-vertex cut.

Observation: $\kappa(G) = \min_{s \neq t} \kappa_G(s,t)$.

Computing a minimum (s, t)-vertex cut

- **Observation:** A minimum (s, t)-vertex cut can be computed in one max-flow call.

Li Chen, Rasmus Kyng, Yang Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva
"Maximum Flow and Minimum-Cost Flow in Almost-Linear Time"
Journal of the ACM, 2025

Although the algorithm is randomized, subsequent work makes it deterministic.

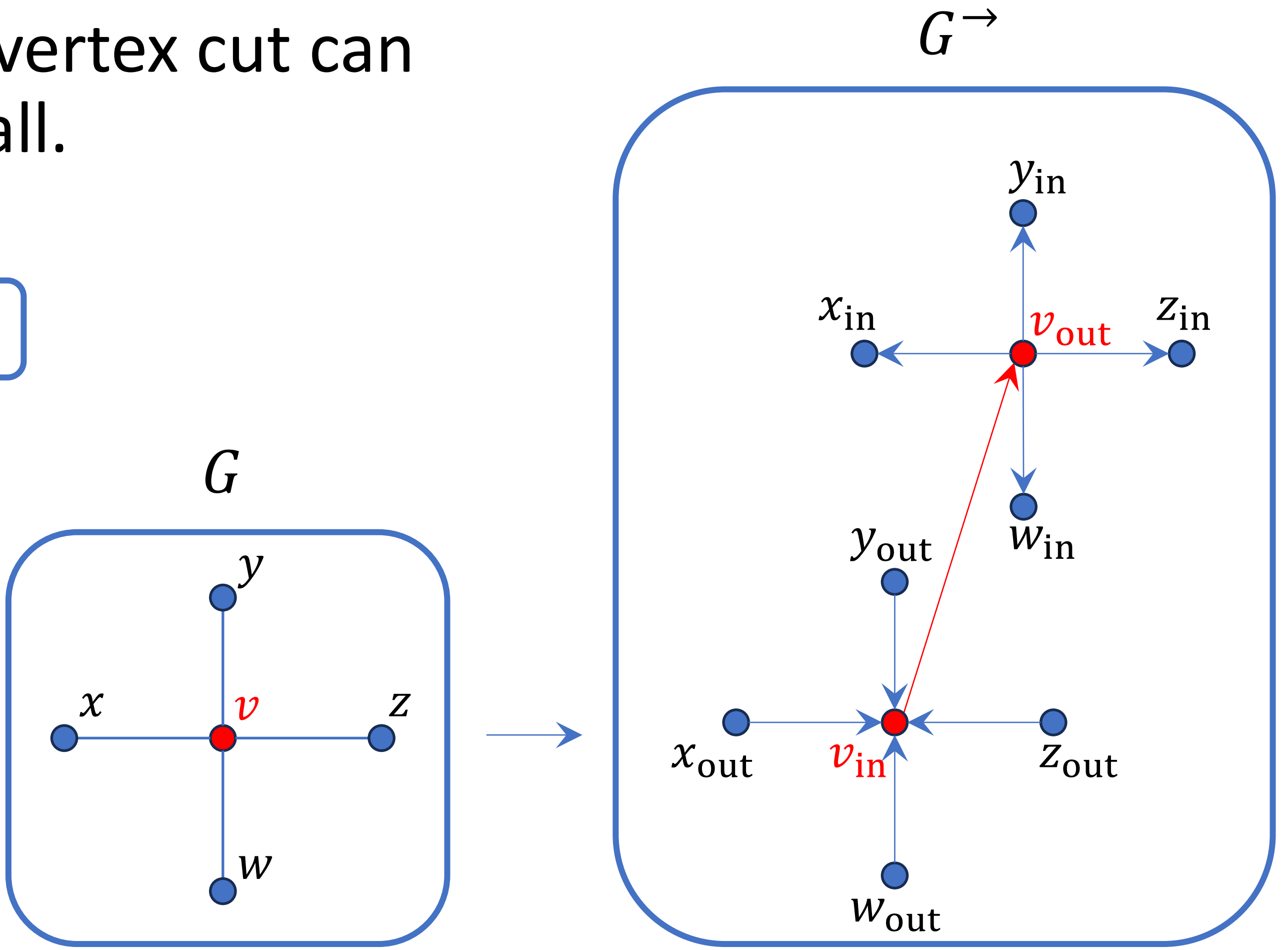
Computing a minimum (s, t)-vertex cut

- **Observation:** A minimum (s, t)-vertex cut can be computed in one max-flow call.

Maximum number of vertex-disjoint (s, t)-paths in G

$=$

Maximum $(s_{\text{out}}, t_{\text{in}})$-flow in $G^{\rightarrow}$



Computing a minimum (s,t)-vertex cut

- **Observation:** A minimum (s,t)-vertex cut can be computed in one max-flow call.

Menger's theorem

Maximum number of vertex-disjoint (s,t)-paths in G = Minimum (s,t)-vertex cut in G

Maximum number of vertex-disjoint (s,t)-paths in G = Maximum $(s_{\text{out}}, t_{\text{in}})$-flow in $G^{\rightarrow}$

Minimum (s,t)-vertex cut in G = Minimum (s,t)-cut in $G^{\rightarrow}$

Maximum $(s_{\text{out}}, t_{\text{in}})$-flow in $G^{\rightarrow}$ = Minimum (s,t)-cut in $G^{\rightarrow}$

Max-flow min-cut theorem

Computing a minimum vertex cut

- **Slow:** Computing a minimum (s,t)-vertex for every pair (s,t).

This requires $O(n^2)$ max-flow calls.

Computing a minimum vertex cut

- **Slow:** Computing a $\underbrace{\text{minimum } (s,t)\text{-vertex for every pair } (s,t)}_{\text{This requires } O(n^2) \text{ max-flow calls.}}$.

- **Key observation:**
  - The problem becomes easier if a minimum vertex cut is β-balanced.

A vertex cut (L, S, R) is $\boldsymbol{\beta}$**-balanced** if $|L| \geq \beta$ and $|R| \in \Omega(n)$.

Computing a minimum vertex cut

- **Slow:** Computing a minimum (s,t)-vertex for every pair (s,t).

This requires $O(n^2)$ max-flow calls.

- **Key observation:**
  - The problem becomes easier if a minimum vertex cut is β-balanced.

A vertex cut (L,S,R) is $\boldsymbol{\beta}$**-balanced** if $|L| \geq \beta$ and $|R| \in \Omega(n)$.

For $\Omega\left(\frac{\beta}{n}\right)$ fraction of the pairs (s,t), we have: $\kappa(G) = \kappa_G(s,t)$.

By computing a minimum (s,t)-vertex cut for $O\left(\frac{n}{\beta}\right)$ random choices of (s,t) and selecting the smallest among them, we obtain a minimum vertex cut with probability at least 0.99.

Can we make it deterministic?

Derandomization via Ramanujan graphs

- We fix a β-balanced minimum vertex cut $\underbrace{(L, S, R)}_{\text{Unknown to the algorithm}}$ of $G = (V, E)$:
  - $|L| \geq \beta$
  - $|R| \in \Omega(n)$
- **Goal:** Select $s \in L$ and $t \in R$.

Derandomization via Ramanujan graphs

- We fix a β-balanced minimum vertex cut $\underbrace{(L, S, R)}_{\text{Unknown to the algorithm}}$ of $G = (V, E)$:
  - $|L| \geq \beta$
  - $|R| \in \Omega(n)$
- **Goal:** Select $s \in L$ and $t \in R$.

Consider a d-regular Ramanujan graph H over the same vertex set V.

Recall: For any $S, T \subseteq V$, if $|S| \cdot |T| \cdot d \geq 4n^2$, then $E_H(S, T) \neq \emptyset$.

$\downarrow$

There is a choice $d \in O\left(\frac{n}{\beta}\right)$ such that:

- If $|L| \geq \beta$ and $|R| \in \Omega(n)$, then $E_H(L, R) \neq \emptyset$.

Derandomization via Ramanujan graphs

- We fix a β-balanced minimum vertex cut $\underbrace{(L, S, R)}_{\text{Unknown to the algorithm}}$ of $G = (V, E)$:
  - $|L| \geq \beta$
  - $|R| \in \Omega(n)$
- **Goal:** Select $s \in L$ and $t \in R$.

$\boxed{\text{Consider a } d\text{-regular Ramanujan graph } H \text{ over the same vertex set } V.}$

Recall: For any $S, T \subseteq V$, if $|S| \cdot |T| \cdot d \geq 4n^2$, then $E_H(S, T) \neq \emptyset$.

There is a choice $d \in O\left(\frac{n}{\beta}\right)$ such that:

- If $|L| \geq \beta$ and $|R| \in \Omega(n)$, then $E_H(L, R) \neq \emptyset$.

It suffices to go over all edges $\{s, t\}$ in H.

This requires $nd \in O\left(\frac{n^2}{\beta}\right)$ max-flow calls.

The balanced case

- **Summary:** If a minimum vertex cut is β-balanced, then a minimum vertex cut can be computed using $O\left(\frac{n^2}{\beta}\right)$ max-flow calls.

Improvement over the naïve $O(n^2)$ bound.

The imbalanced case

- **Summary:** If a minimum vertex cut is β-balanced, then a minimum vertex cut can be computed using $O\left(\frac{n^2}{\beta}\right)$ max-flow calls.

Improvement over the naïve $O(n^2)$ bound.

- **Next:** Extend the approach to the imbalanced case.

The plan: gradually modify the graph to make it balanced.

The algorithm works under the condition: $\delta(G) \leq 0.99 \cdot n$

The $\delta - \kappa$ gap

- We write $\delta(G)$ to denote the minimum degree of G.

Claim: Every minimum vertex cut (L^*, S^*, R^*) of $G = (V, E)$ is β-balanced for $\beta = \underbrace{\delta(G) - \kappa(G)}$.

We always have: $\delta(G) \geq \kappa(G)$

The algorithm works under the condition: $\delta(G) \leq 0.99 \cdot n$

The $\delta - \kappa$ gap

- We write $\delta(G)$ to denote the minimum degree of G.

Claim: Every minimum vertex cut (L^*, S^*, R^*) of $G = (V, E)$ is β-balanced for $\beta = \delta(G) - \kappa(G)$.

By symmetry, assume $|L^*| \leq |R^*|$.

$$|R^*| \geq \frac{n - |S^*|}{2} = \frac{n - \kappa(G)}{2} \geq \frac{n - \delta(G)}{2} \geq \frac{0.01 \cdot n}{2}$$

The algorithm works under the condition: $\delta(G) \le 0.99 \cdot n$

The $\delta - \kappa$ gap

- We write $\delta(G)$ to denote the minimum degree of G.

Claim: Every minimum vertex cut (L^*, S^*, R^*) of $G = (V, E)$ is β-balanced for $\beta = \delta(G) - \kappa(G)$.

By symmetry, assume $|L^*| \le |R^*|$.

$$|R^*| \ge \frac{n - |S^*|}{2} = \frac{n - \kappa(G)}{2} \ge \frac{n - \delta(G)}{2} \ge \frac{0.01 \cdot n}{2}$$

Consider any $v \in L^*$.

$$\delta(G) \le \deg(v) < |L^*| + |S^*| = |L^*| + \kappa(G) \quad \rightarrow \quad |L^*| > \delta(G) - \kappa(G) = \beta$$

The algorithm works under the condition: $\delta(G) \leq 0.99 \cdot n$

Gap amplification

- We write $\delta(G)$ to denote the minimum degree of G.

Claim: Every minimum vertex cut (L^*, S^*, R^*) of $G = (V, E)$ is β-balanced for $\beta = \delta(G) - \kappa(G)$.

The plan:
We will show that $O(n)$ max-flow calls suffice to achieve one of the following.

- A minimum vertex cut.
- Increasing the $\delta - \kappa$ gap by 1.

The algorithm works under the condition: $\delta(G) \leq 0.99 \cdot n$

Gap amplification

- We write $\delta(G)$ to denote the minimum degree of G.

Claim: Every minimum vertex cut (L^*, S^*, R^*) of $G = (V, E)$ is β-balanced for $\beta = \delta(G) - \kappa(G)$.

The plan:

We will show that $O(n)$ max-flow calls suffice to achieve <u>one of the following</u>.

- A minimum vertex cut.
- Increasing the $\delta - \kappa$ gap by 1.

After repeating this for β times, we are in the β-balanced case.

This takes $O(\beta n)$ max-flow calls.

A minimum vertex cut can be computed using $O\left(\frac{n^2}{\beta}\right)$ max-flow calls.

The algorithm works under the condition: $\delta(G) \leq 0.99 \cdot n$

Gap amplification

- We write $\delta(G)$ to denote the minimum degree of G.

Claim: Every minimum vertex cut (L^*, S^*, R^*) of $G = (V, E)$ is β-balanced for $\beta = \delta(G) - \kappa(G)$.

The plan:

We will show that $O(n)$ max-flow calls suffice to achieve <u>one of the following</u>.

- A minimum vertex cut.
- Increasing the $\delta - \kappa$ gap by 1.

Total cost: $O(n^{1.5})$ max-flow calls.

$\beta = \sqrt{n}$

After repeating this for β times, we are in the β-balanced case.

This takes $O(\beta n)$ max-flow calls.

A minimum vertex cut can be computed using $O\left(\frac{n^2}{\beta}\right)$ max-flow calls.

Single-source vertex connectivity

- **Define:** $\underbrace{\kappa_G(s) = \min_{t \neq s} \kappa_G(s,t).}$

Can be computed using $n - 1$ max-flow calls.

Single-source vertex connectivity

- **Define:** $\underbrace{\kappa_G(s) = \min_{t \neq s} \kappa_G(s,t).}_{\text{Can be computed using } n-1 \text{ max-flow calls.}}$

Claim: If $\kappa_G(s) > \kappa(G)$, then $s \in S^*$ for every minimum vertex cut (L^*, S^*, R^*) of G.

If $s \notin S^*$, then $|S^*| \geq \kappa_G(s) > \kappa(G)$, so (L^*, S^*, R^*) is not a minimum cut.

Algorithm

- Select a vertex s.
- Compute a minimum (s,t)-vertex cut for every $t \neq s$.
- **Case 1:** $\kappa_G(s) = \kappa(G)$.
  - We have already obtained a minimum vertex cut.
- **Case 2:** $\kappa_G(s) > \kappa(G)$.

Algorithm

- Select a vertex s.
- Compute a minimum (s, t)-vertex cut for every $t \neq s$.
- **Case 1:** $\kappa_G(s) = \kappa(G)$.
  - We have already obtained a minimum vertex cut.
- **Case 2:** $\kappa_G(s) > \kappa(G)$.
  - $s \in S^*$ for every minimum vertex cut (L^*, S^*, R^*) of G.
  - Consider $G' = G - s$.

$\boxed{(L^*, S^*, R^*) \text{ is a minimum cut of } G} \longleftrightarrow \boxed{(L^*, S^* \setminus \{s\}, R^*) \text{ is a minimum cut of } G'}$

This reduces the problem of computing a minimum cut in G to the same problem in G'.

Algorithm

Clarification: Since the algorithm cannot determine whether it is in Case 1 or Case 2, it records the cuts obtained in both cases. The smaller of the two is guaranteed to be a minimum cut.

- Select a vertex s.
- Compute a minimum (s, t)-vertex cut for every $t \neq s$.
- **Case 1:** $\kappa_G(s) = \kappa(G)$.
  - We have already obtained a minimum vertex cut.
- **Case 2:** $\kappa_G(s) > \kappa(G)$.
  - $s \in S^*$ for every minimum vertex cut (L^*, S^*, R^*) of G.
  - Consider $G' = G - s$.

(L^*, S^*, R^*) is a minimum cut of G $\longleftrightarrow$ $(L^*, S^* \setminus \{s\}, R^*)$ is a minimum cut of G'

This reduces the problem of computing a minimum cut in G to the same problem in G'.

Algorithm

Recall: We want to increase the $\delta - \kappa$ gap by 1.

- Since $\kappa(G') = \kappa(G) - 1$, we achieve the goal if $\delta(G') = \delta(G)$.
- However, it is possible that $\underbrace{\delta(G') = \delta(G) - 1 \text{ due to the removal of vertex } s}_{\text{How to fix it?}}$.

- Select a vertex s.
- Compute a minimum (s, t)-vertex cut for every $t \neq s$.
- **Case 1:** $\kappa_G(s) = \kappa(G)$.
  - We have already obtained a minimum vertex cut.
- **Case 2:** $\kappa_G(s) > \kappa(G)$.
  - $s \in S^*$ for every minimum vertex cut (L^*, S^*, R^*) of G. $\longrightarrow$ $\kappa(G') = \kappa(G) - 1$
  - Consider $G' = G - s$.

$\boxed{(L^*, S^*, R^*) \text{ is a minimum cut of } G} \longleftrightarrow \boxed{(L^*, S^* \setminus \{s\}, R^*) \text{ is a minimum cut of } G'}$

This reduces the problem of computing a minimum cut in G to the same problem in G'.

Degree restoration

- **Setting:**
  - $G = (V, E)$.
  - $G' = G - s$.
  - $F = \{v \in V : \{v, s\} \in E,\ \deg_G(v) = \delta(G)\}$

    - This is the set of vertices whose degrees drop to $\delta(G) - 1$ after the removal of s.
    - If we can restore their degrees to $\delta(G)$, then the goal is achieved.
    - The challenge is to do so in a way that does not affect the minimum vertex cut.

Degree restoration

How to see the existence of u?

- **Reason 1:** The assumption that the minimum degree of the input graph is at most $0.99 \cdot n$.
- **Reason 2:** If u does not exist, then G' is already a clique.

- **Setting:**
  - $G = (V, E)$.
  - $G' = G - s$.
  - $F = \{v \in V : \{v, s\} \in E,\ \deg_G(v) = \delta(G)\}$

- For each $v \in F$, select a vertex $u \in V$ such that $\{u, v\} \notin E$.
- Compute a minimum (u, v)-vertex cut in G'.
- **Case 1:** $\kappa_{G'}(u, v) = \kappa(G')$.
  - We have already obtained a minimum vertex cut in G'.
- **Case 2:** $\kappa_{G'}(u, v) > \kappa(G')$.

Degree restoration

$\kappa_{G'}(u, v) > \kappa(G')$

- **Setting:**
  - $G = (V, E)$.
  - $G' = G - s$.
  - $F = \{v \in V : \{v, s\} \in E,\ \deg_G(v) = \delta(G)\}$

For every minimum cut (L^*, S^*, R^*) of G', we have:

- $\{u, v\} \subseteq L^* \cup S^*$ or $\{u, v\} \subseteq R^* \cup S^*$.

- For each $v \in F$, select a vertex $u \in V$ such that $\{u, v\} \notin E$.
- Compute a minimum (u, v)-vertex cut in G'.
- **Case 1:** $\kappa_{G'}(u, v) = \kappa(G')$.
  - We have already obtained a minimum vertex cut in G'.
- **Case 2:** $\kappa_{G'}(u, v) > \kappa(G')$.
  - We have $\kappa(G') = \kappa(G' + \{u, v\})$.
  - Therefore, we may set: $G' \leftarrow G + \{u, v\}$.

This restore v's degree to $\delta(G)$

Degree restoration

Clarification: Since the algorithm cannot determine whether it is in Case 1 or Case 2, it records the cuts obtained in both cases. The smaller of the two is guaranteed to be a minimum cut.

- **Setting:**
  - $G = (V, E)$.
  - $G' = G - s$.
  - $F = \{v \in V : \{v, s\} \in E,\ \deg_G(v) = \delta(G)\}$

- For each $v \in F$, select a vertex $u \in V$ such that $\{u, v\} \notin E$.
- Compute a minimum (u, v)-vertex cut in G'.
- **Case 1:** $\kappa_{G'}(u, v) = \kappa(G')$.
  - We have already obtained a minimum vertex cut in G'.
- **Case 2:** $\kappa_{G'}(u, v) > \kappa(G')$.
  - We have $\kappa(G') = \kappa(G' + \{u, v\})$.
  - Therefore, we may set: $G' \leftarrow G + \{u, v\}$.

This restore v's degree to $\delta(G)$

References

- **Main reference:**
  - Lecture 5.2 of https://sites.google.com/site/thsaranurak/teaching/Expander

- **Additional/optional reading:**
  - Harold N. Gabow. "Using expander graphs to find vertex connectivity." Journal of the ACM (2006) https://doi.org/10.1145/1183907.1183912

The algorithm presented in this lecture is from this paper.