CS6130: Advanced Graph Algorithms

Haricharan & Abhinav

Vital Edges for $(s, t)-$ mincut: Efficient Algorithms, Compact Structures, and Optimal Sensitivity Oracle

About the paper

Some general remarks!

- Surender Baswana, Koustav Bhanja (IITK)
- Published in arXiv publication in 2023
- Contains:
  - Generalization of max-flow mincut theorem
  - Computation of vital edges
  - Efficient data structures for storing mincuts of vital edges
  - Sensitivity oracle for online updates of the graph and mincut queries

Overview of the Presentation

We'll cover all these today!

- What are Vital edges?
- Generalization of Maxflow-mincut theorem for an edge
- Vital edges: tight and loose edges
- Computing the tight edges
- Computing the loose edges

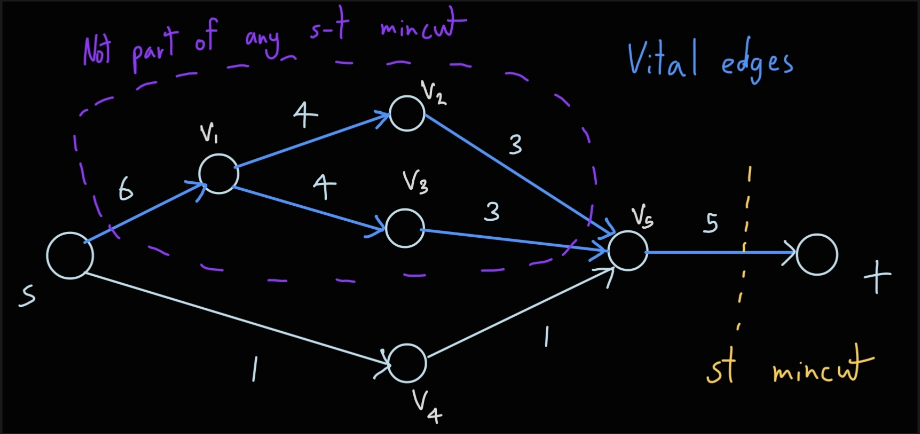
We have a directed graph $G(V, E)$ with edge weights $w(e) : E \to \mathbb{R}$, and two vertices s and t.

Vital Edge

An edge $e \in E$ is said to be vital if removing e decreases the capacity of the s-t mincut. The vitality of an edge ($w_{min}(e)$) is the reduction in the capacity of the s-t mincut on removal of e.

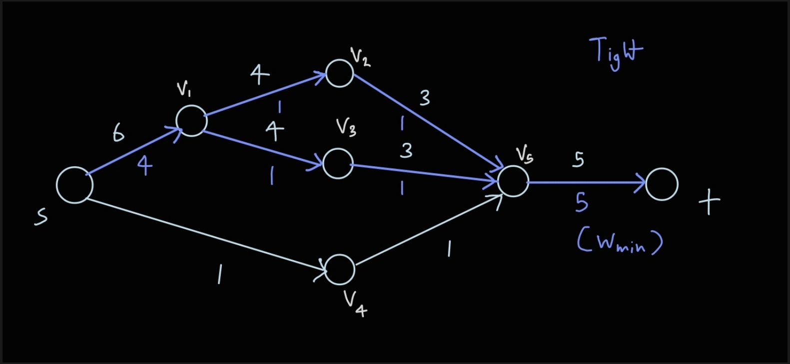
Note: w_{min} is called so because it is the minimum flow through the edge in any s-t maxflow. Alternatively, it can be interpreted as the *minimum weight* the edge can have without affecting the value of the s-t mincut.

Example of Vital edges



Note: Every edge in a mincut is a vital edge, but there are more vital edges!

Example of Vital edges



A Small Lemma

We'll be using this later!

Lemma

An edge e is a vital edge if and only if $f(e) > 0$ in every maximum (s, t)-flow in G.

Forward Direction (by contrapositive):

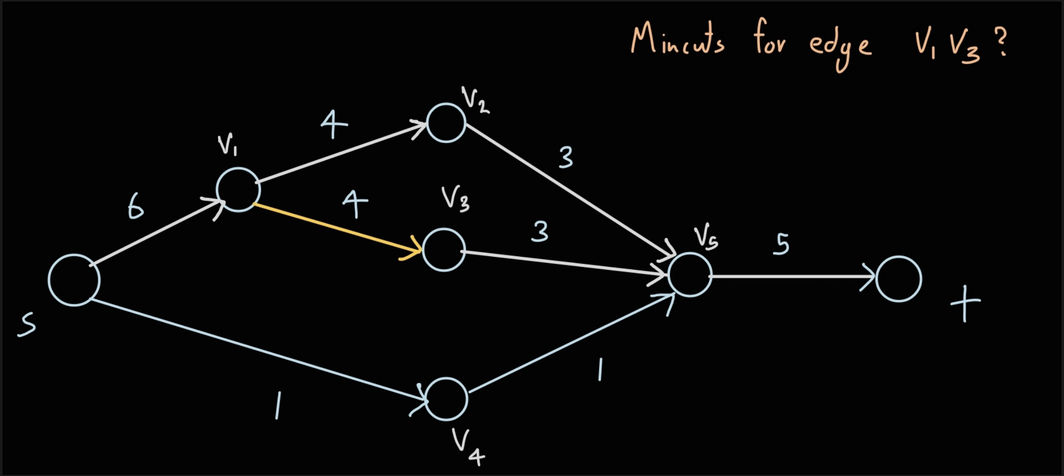
Let $f(e) = 0$ in some maxflow F.

Now, consider the same flow F in the graph $G(V, E \setminus \{e\})$, and clearly this is a valid flow.

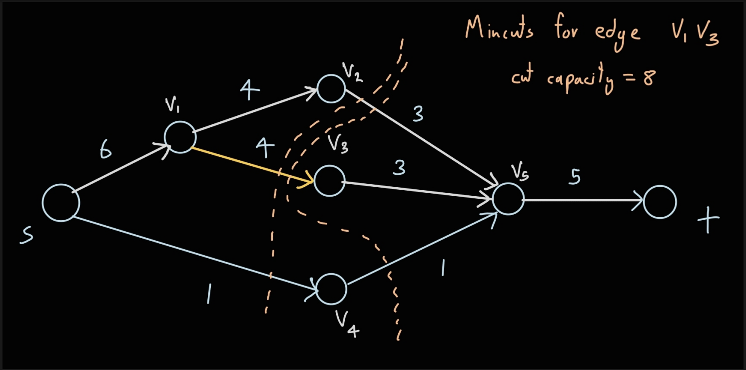
By maxflow-mincut theorem, the value of the mincut also remains the same, and hence e is non-vital.

Backward direction: Similar proof.

Mincut for an edge?



Mincut for an edge

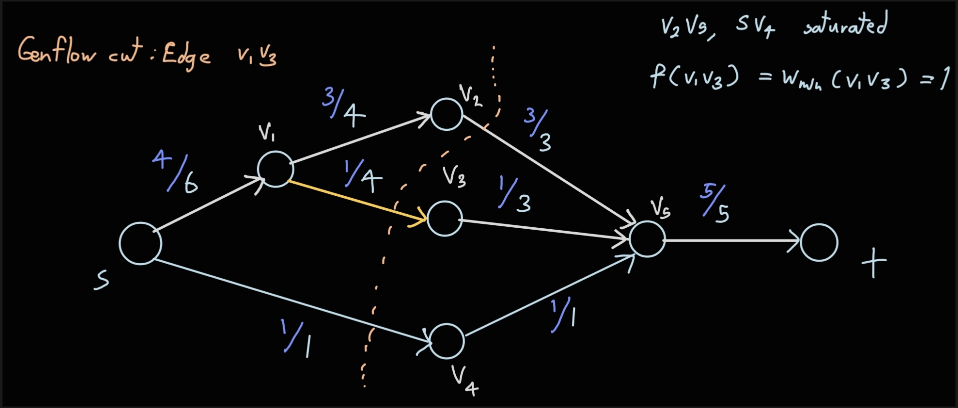


Among all the s-t cuts containing a particular edge e, the one with the minimum capacity is called the mincut for that edge.

Denoted by $C(e)$.

Convention: $C(e)$ is a set of vertices, *not edges*.

An Interesting Observation

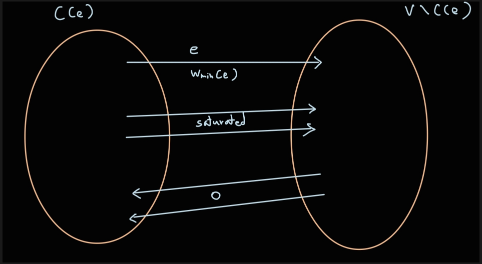


Generalization of Maxflow-mincut theorem for vital edges

GenFlowCut Theorem

Consider a mincut for a vital edge e, $C(e)$. $C(e)$ is a mincut for e iff there is a maximum s-t flow such that:

- $f_{in}(C(e)) = 0$
- Outgoing edges in $C(e) \setminus \{e\}$ are saturated
- $f(e) = w_{min}(e)$, where $w_{min}(e)$ is the amount by which mincut value decreases on e's removal.



Proof

The forward direction is similar!

Backward Direction:

$C(e)$ is a cut for an edge e in graph G. Consider the maxflow f^* in G with the properties mentioned before.

$f^* = c_G(C(e)) - w(e) + w_{min}(e)$, or $\boxed{c_G(C(e)) = f^* + w(e) - w_{min}(e)}$.

Now, $c_{G \setminus \{e\}}(C(e)) = f^* - w_{min}(e)$ (this comes from the definition itself).

$\implies C(e)$ is a s-t mincut in $G \setminus \{e\}$.

Hence, each s-t cut to which e contributes has capacity at least $f^* - w_{min}(e) + w(e)$ in G, and hence $C(e)$ is a mincut for the edge e!

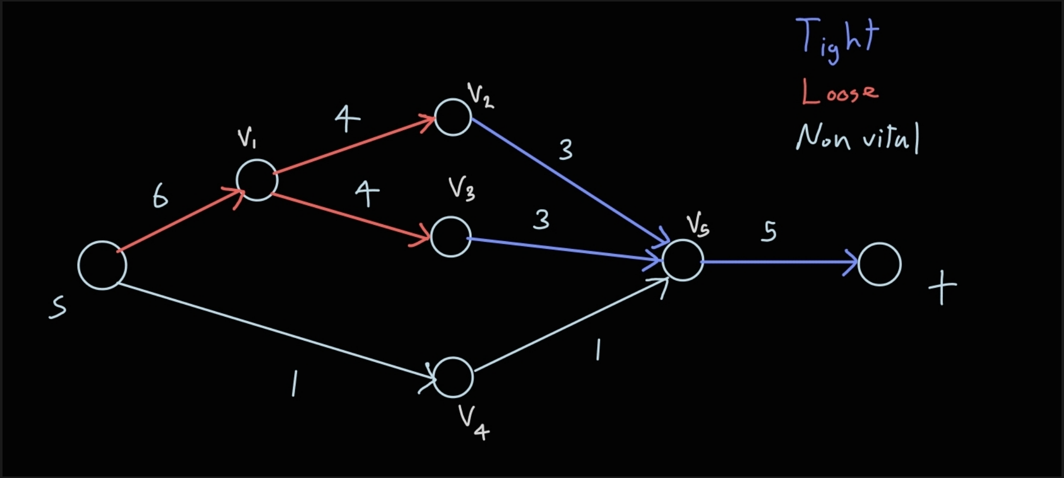
Computing all vital edges

We can do this naively in $O(m)$ flow computations, by just removing each edge from the graph and finding the maxflow. But, we can do this in $\boxed{O(n)}$ flow computations actually!

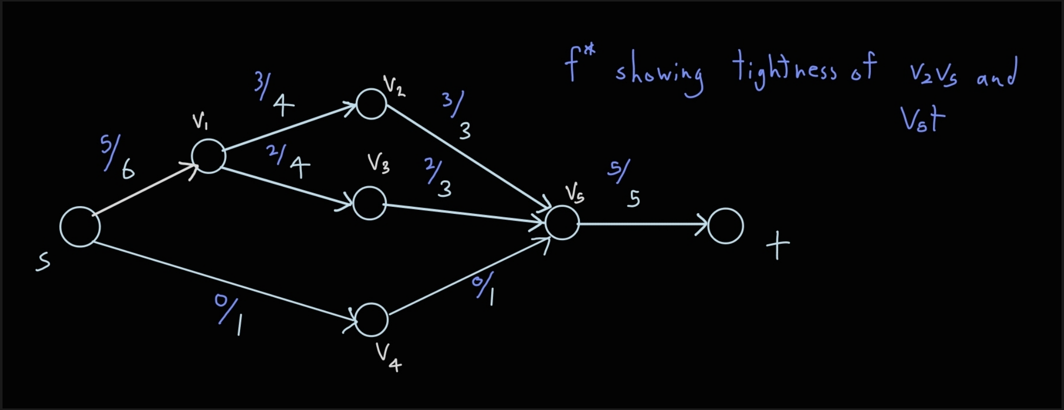
Vital edges are characterized as:

- **Tight edge:** A vital edge e is tight iff $\exists$ a maximum s-t flow which saturates e
- **Loose edge:** Otherwise

Example of Tight and Loose Edges



Example of Tight and Loose Edges



Computing Loose edges:

There's a way to compute loose edges using $O(n)$ flow computations. This is not very illustrative for our purposes, so this method can be found at the end of our slides.

Computing Tight edges:

- We will use the ancestor tree (explained later).
- There is a way to get LCA in $O(1)$ time (the usually way using binary lifting is $O(\log n)$).
- The LCA in the ancestor tree gives a mincut for a particular edge, and of course the value of the mincut.

Ancestor Tree $\mathcal{T}$ (Cheng and Hu 1991)

Given a cost function for cuts $F(C)$ (C is taken to be a set of vertices), the Ancestor Tree answers the question: "Given two vertices u, v, what is the cut of minimum cost separating u and v?"

Preprocessing: $O(n)$ min-cut computations (Note that these are not standard max-flow mincut computations, but are of the form "Find the cut with minimum $F(C)$ separating u and v")

Time complexity: $O(1)$ per query (if only capacity required) or $O(|C|)$ per query (To return the cut C)

Space complexity: $O(n)$ if only capacities needed, $O(n^2)$ if cuts are also needed

Using the ancestor tree to find vital edges

Suppose edge (u, v) is vital. Then by GenFlowCut, $\exists$ s-t cut C in which (u, v) contributes and $c(C) = f^* - w_{min}(u, v) + w(u, v)$.

The minimum capacity s-t cut C' that separates u and v has $c(C') \leq c(C)$ which implies

$$c(C') \leq f^* - w_{min}(u, v) + w(u, v)$$

$$c(C') < f^* + w(u, v) \qquad \text{(Vital: } w_{min}(u, v) > 0\text{)}$$

$$\implies c(C') - w(u, v) < f^*$$

Using the ancestor tree to find vital edges

Let $F : C \subset V \to \mathbb{R}$, $F(C) = \begin{cases} c(C), s \in C, t \in \bar{C} \\ \infty, \text{ otherwise} \end{cases}$

If C is an s-t cut, then $F(C)$ is its capacity. Otherwise, $F(C)$ is infinite, ensuring all the cuts returned are s-t cuts.

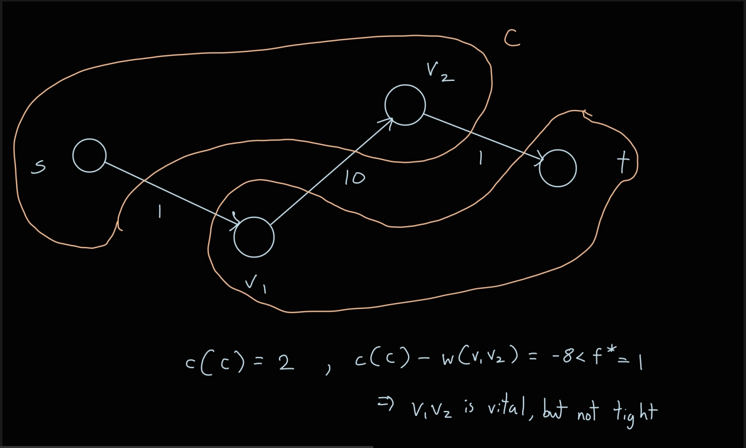
Now we can iterate over all edges (u, v). If the cut C separating (u, v) in the tree satisfies $c(C) - w(u, v) < f^*$ then the edge is vital. Since we already know the loose vital edges, we take all the other vital edges to be tight.

Since the ancestor tree allows $O(1)$ queries for the minimum separating cut capacity, we now have an $O(m)$ algorithm $(+O(n)$ mincut computations initially).

Ancestor Tree - Caution

The cut stored in the tree is not necessarily the mincut for that edge!

Example of an edge $(v_1 v_2)$ where the minimum s-t cut separating its vertices is NOT the same as the mincut of that edge:

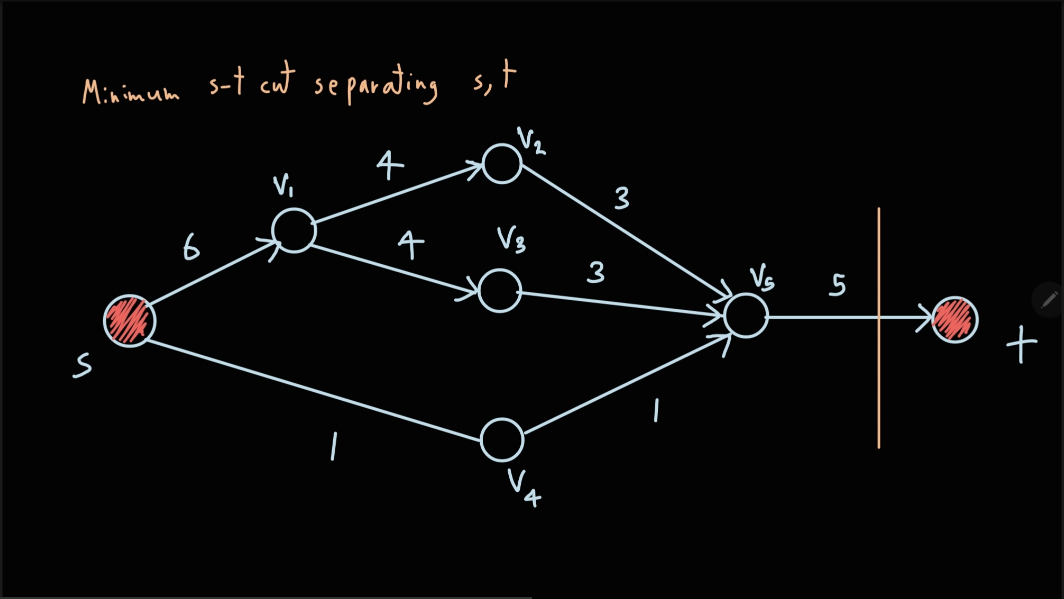


Ancestor Tree - Example of Construction

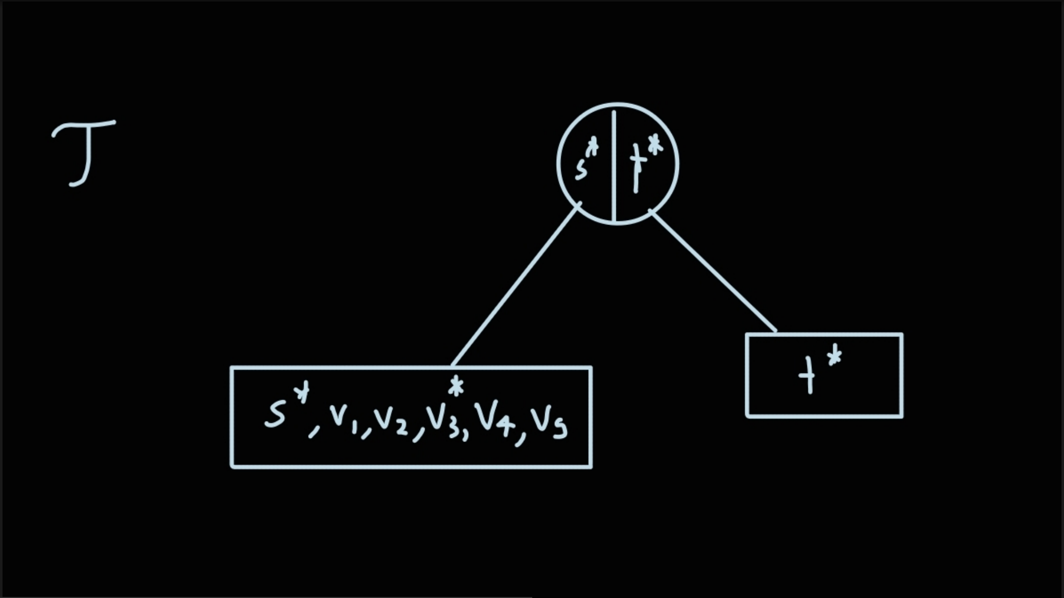
T

$\boxed{s^*, v_1, v_2, v_3, v_4, v_5, t^*}$

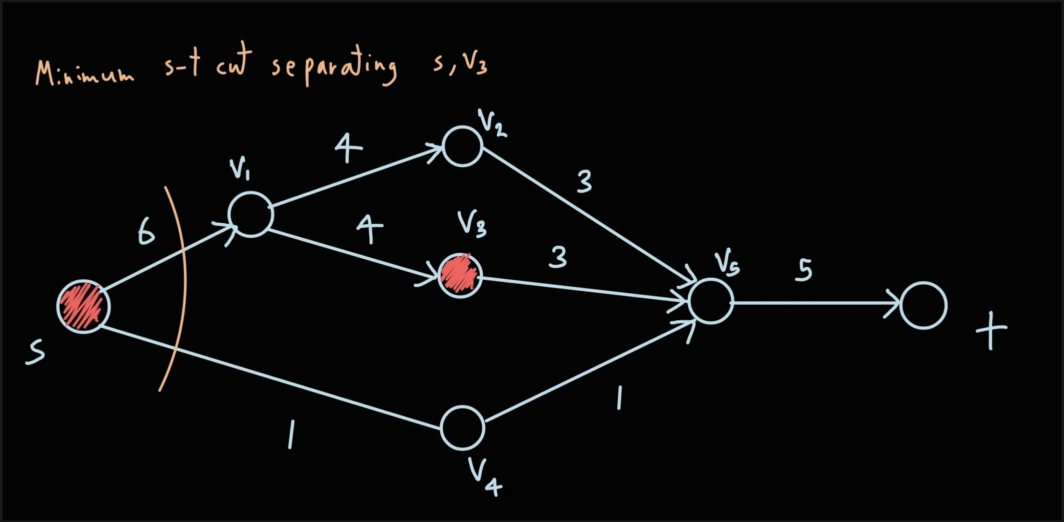
Ancestor Tree - Example of Construction



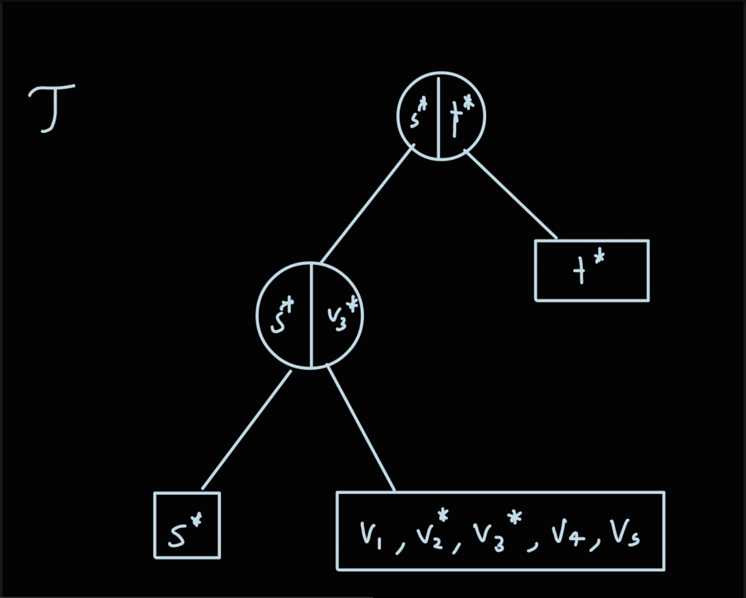
Ancestor Tree - Example of Construction



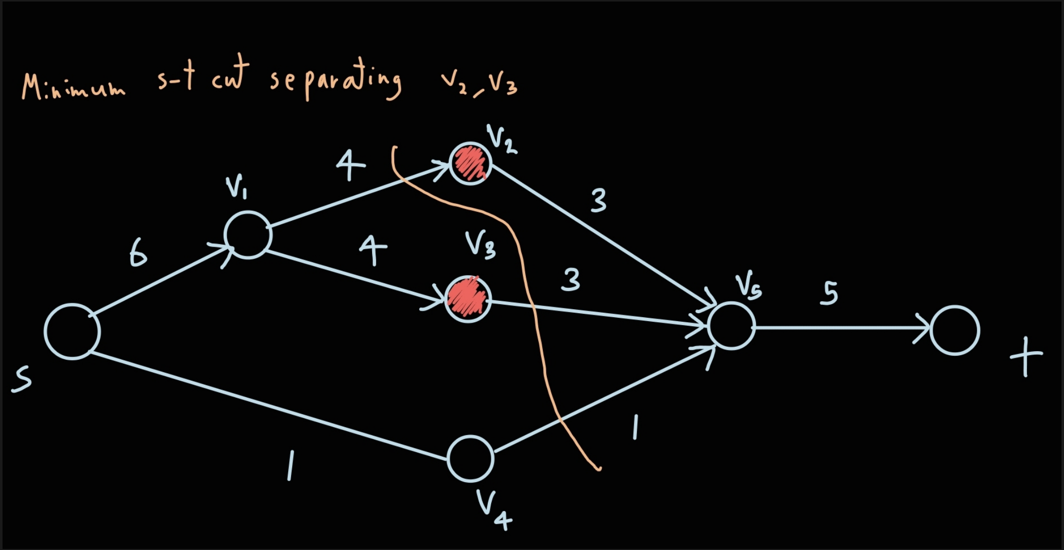
Ancestor Tree - Example of Construction



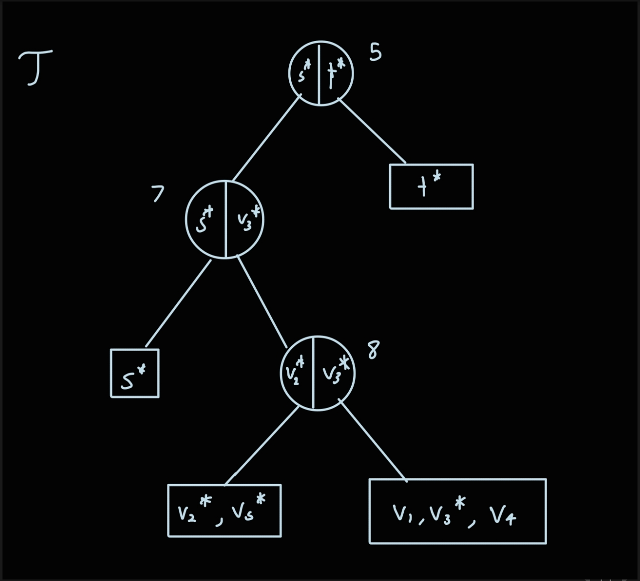
Ancestor Tree - Example of Construction



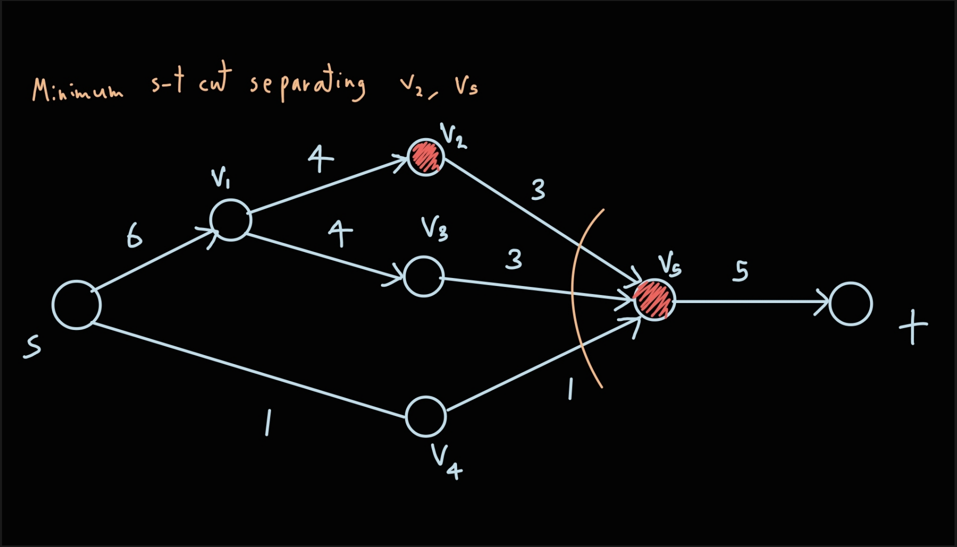
Ancestor Tree - Example of Construction



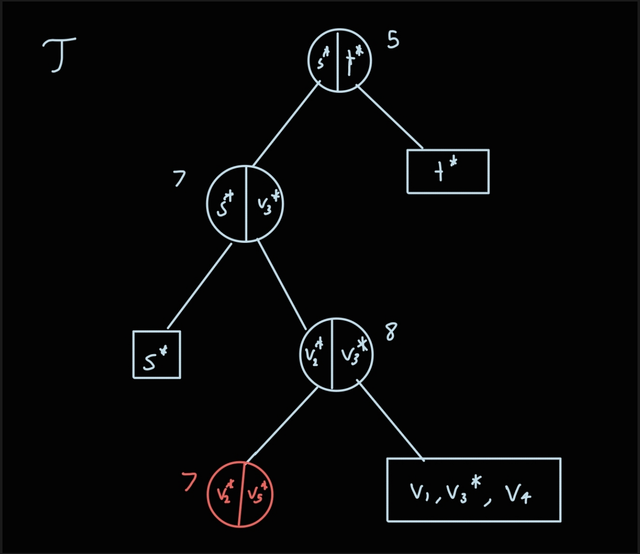
Ancestor Tree - Example of Construction



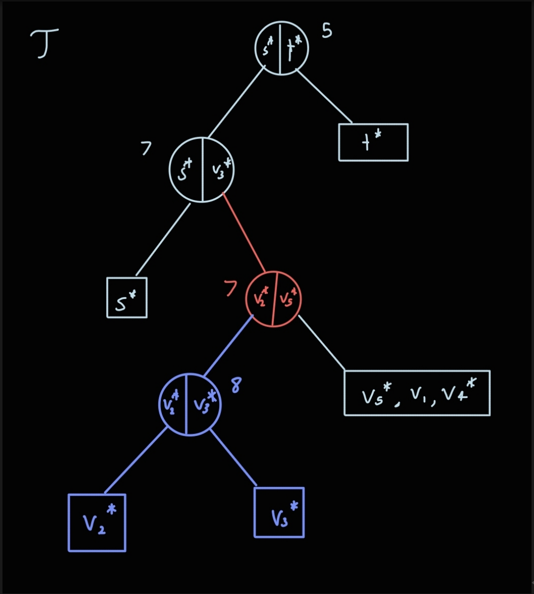
Ancestor Tree - Example of Construction



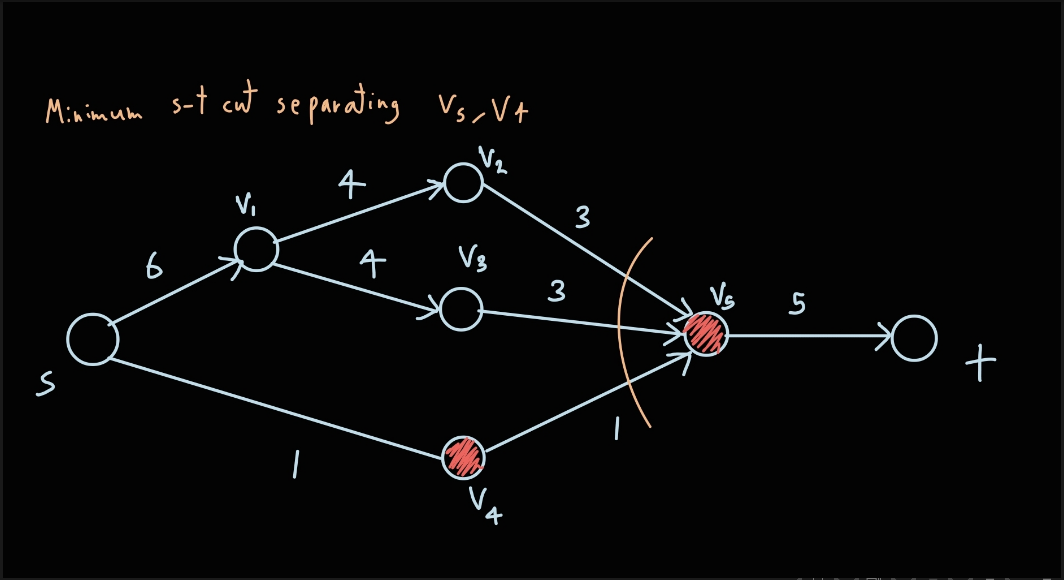
Ancestor Tree - Example of Construction



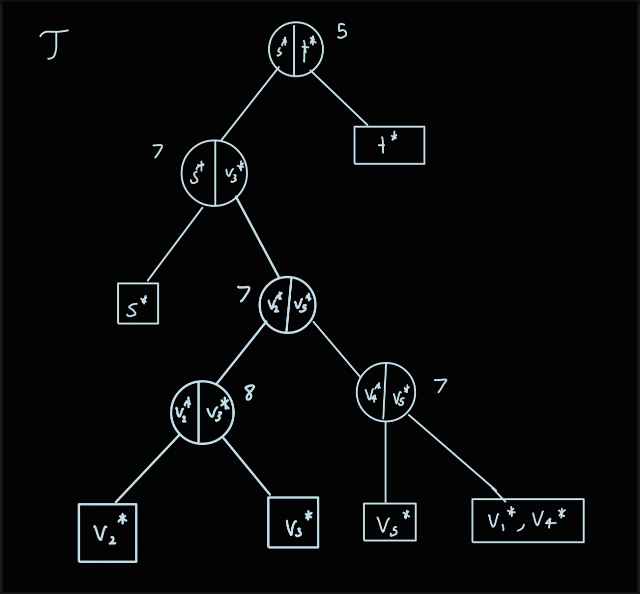
Ancestor Tree - Example of Construction



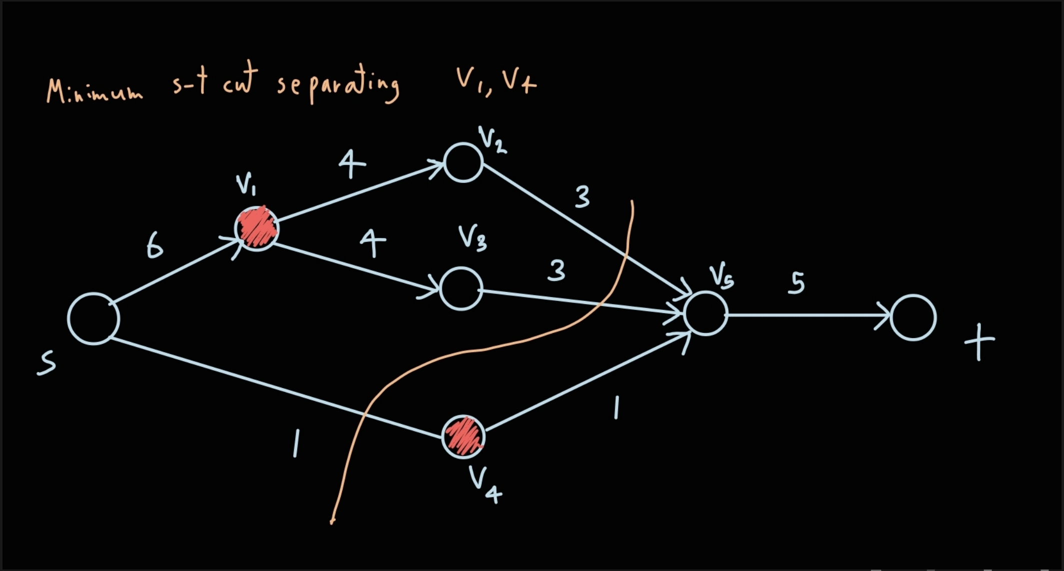
Ancestor Tree - Example of Construction



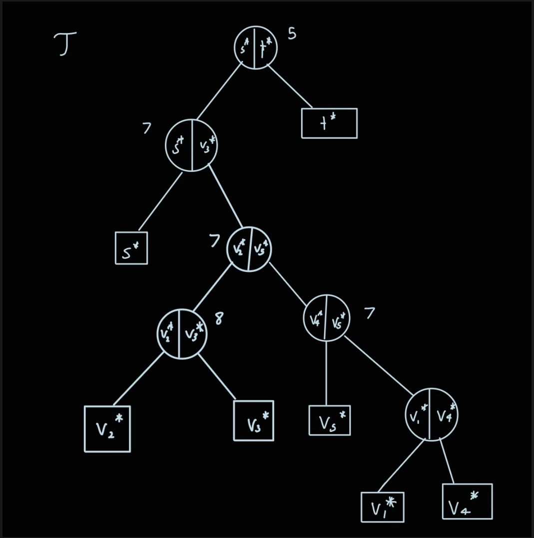
Ancestor Tree - Example of Construction



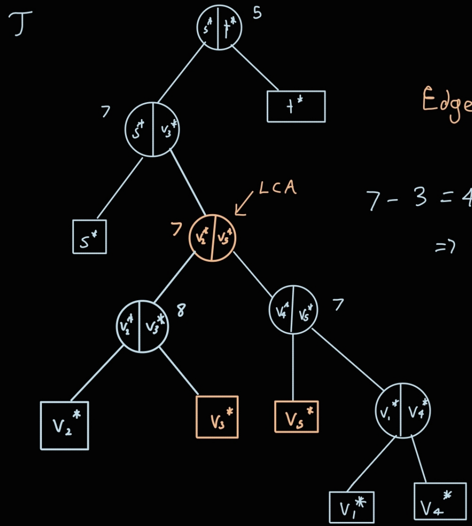
Ancestor Tree - Example of Construction



Ancestor Tree - Example of Construction



Ancestor Tree - Example of Query

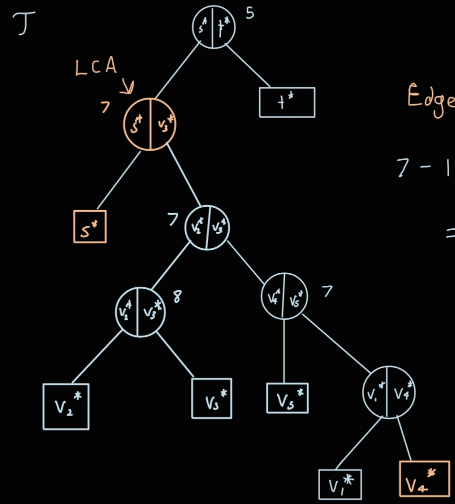


Edge $v_3 v_5$

$7 - 3 = 4 < 5$

$\Rightarrow v_3 v_5$ is vital !

Ancestor Tree - Example of Query



Edge sV_4

$7 - 1 = 6 \not< 5$

$\Rightarrow sV_4$ is not vital

Ancestor Tree - Summary

Each internal node of the ancestor tree has 3 properties

- Two special 'seeded' vertices u, v
- The capacity of a minimum cut C separating u and v

And each leaf of the tree has 1 property:

- A set of vertices $V' \subseteq V(G)$

Each internal node has two children (and thus two child subtrees): One such subtree contains vertices in C and the other contains vertices in $V' \setminus C$.

The tree is constructed so that the capacity of minimum cuts increases as we go down from the root.

The minimum capacity cut seperating any two vertices u and v is stored in the LCA of leaves containing u and v.

Ancestor Tree - Proofs

Intuition for why the process takes only $n - 1$ mincut computations:

- $\mathcal{T}$ starts with only 1 leaf
- Each mincut computation lets us add exactly 1 more leaf
- When the construction ends, there are n leaves

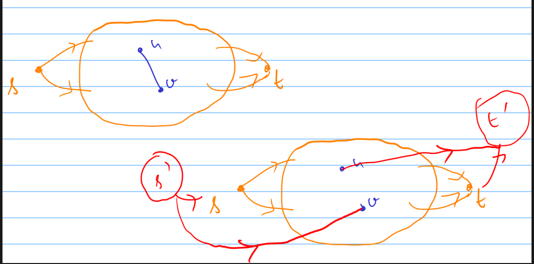
Computing loose edges:

For any directed weighted graph G, there exists a maximum (s, t)-flow $f^{\#}$ in G such that the number of edges that carry nonzero flow but are not fully saturated is at most $n-1$. These are "candidates" for loose edges, we can check what is the maximum flow they can carry. This can be done by the algorithm.

Claim:

We have black box algorithm which gives maximum flow through an edge in any maxflow.

The reduction:



The algorithm

- There is a maxflow in G' which carries f^* amount of flow through e_s and e_t.
- Maxflow in G' is $f^* + \alpha$ implies and implied by α is the maximum flow through e in any maxflow.

Key Takeaways

- There's an equivalent of maxflow-mincut theorem for all vital edges (GenFlowCut)
- We can compute all tight edges using the ancestor tree data structure
- We can compute all loose edges by seeing which of them are candidates and just checking each of them

Other things in the paper

- $O(m)$ space DAG partial characterization of all s-t mincuts
- $O(mn)$ space complete characterization of all s-t mincuts
- Sensitivity Oracles: Online updates to edge weights and asking vitality queries

The QR: