# COL751 - Lecture 17

## 1 An $O(m+n)$ time construction for $k$-edge connectivity preserver

Let $G = (V, E)$ be a multigraph on $n$ vertices and $m$ edges [1]. Recall that a trivial algorithm to compute a $k$-edge connectivity preserver $H$ involves computation of $k$ forests $T_1, \ldots, T_k$ such that $T_i$ is spanning forest of graph $G - \big(E(T_1) \cup \cdots \cup E(T_{i-1})\big)$, for $i \in [1, k]$. Finally, we set $H = (V, E_H)$ where $E_H = \cup_{i \leqslant k} E(T_i)$ is the union of the edges of $k$ forests. The time complexity of this algorithm is $O(n + mk)$.

We will show how to compute in $O(m + n)$ time trees $T_1, \ldots, T_m$ that satisfy the relation that $T_i$ is a spanning forest of graph $G - (T_1 \cup \cdots \cup T_{i-1})$, for each $i \in [1, m]$.

In our algorithm, we compute a mapping $L : V \cup E \to [1, m]$ such that the label $L(e)$ of an edge indicates the index of forest in which $e$ would be contained. The mapping $L$ would satisfy the following invariant throughout the algorithm run.

**Invariant 1** *Let $V_0$ be set of vertices that satisfy the condition that at least one edge incident to them is not assigned a non-zero label. Then, for each $x \in V$:*

$$\{L(e) \mid e \text{ is incident to } x\} = \begin{cases} [0,\ L(x)] & if \quad x \in V_0, \\ [1,\ L(x)] & if \quad x \notin V_0. \end{cases}$$

**foreach** $x \in V \cup E$ **do** $L(x) = 0$.
Initialize $V_0$ as $V$.
**while** $|V_0| > 0$ **do**
    Choose a vertex $x \in V_0$ maximizing $L(x)$.
    **foreach** edge $e = (x, y)$ incident to $x$ such that $L(e) = 0$ **do**
        $L(e) = \min\{L(x), L(y)\} + 1$.
        **if** $L(x) < L(e)$ **then** $L(x) = L(x) + 1$.
        **if** $L(y) < L(e)$ **then** $L(y) = L(y) + 1$.
        Remove $x$ (resp. $y$) from $V_0$ if it has no edges incident of label 0.
Return $L$.

**Algorithm 1:** Computation of mapping $L$ from edges of $G$ to trees $T_1, \ldots, T_m$.

**Lemma 1** *Throughout the algorithm run Invariant 1 holds.*

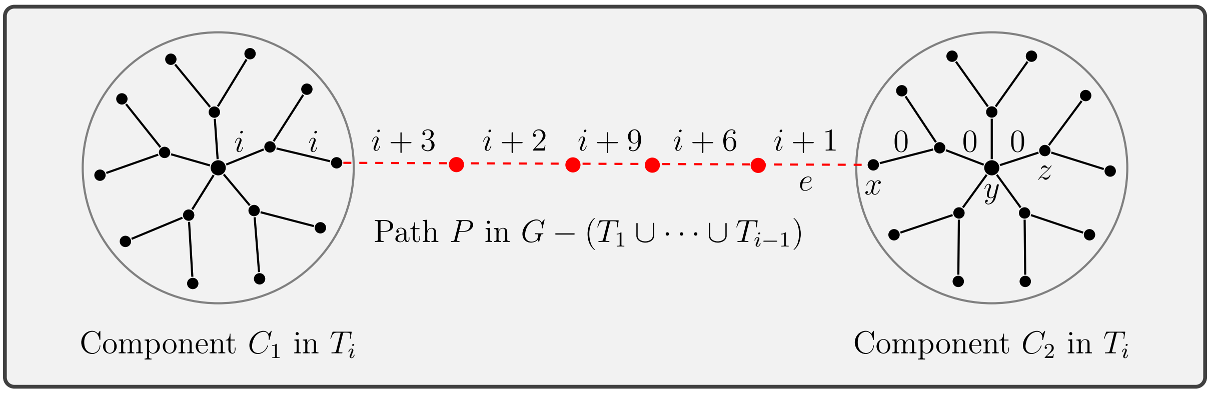
[1] As $G$ is a multigraph $m$ can be much larger than $n^2$.

**Proof:** Suppose the invariant holds before assigning label $L(e)$ to an edge $e = (x, y)$. So the labels of edges incident to $x$ lie in range $[0, L(x)]$, and the labels of edges incident to $y$ lie in range $[0, L(y)]$.

Let us suppose $L(x) = i$ is at least $L(y) = j$. Then after updating label of $e$, we have $L(e) = j + 1$ and $L(y) = j + 1$. Also $L(x)$ is incremented by 1 if updated label $L(e)$ is greater than $L(x)$. Finally, observe that we remove $x$ (resp. $y$) from $V_0$ if they have no edges incident of label 0. Thus, the invariant holds true throughout algorithm run. □

**Lemma 2** *For $i \geqslant 1$, graph $T_i$ is acyclic.*

**Proof:** Assume on contrary there is a cycle $C = (x_1, x_2, \ldots, x_\ell, x_1)$ in $T_i$, and let us suppose $e = (x_1, x_\ell)$ is the edge in $C$ that is labeled last. Then, by Lemma 1 we have that before updating $L(e)$ the labels $L(x_1), L(x_\ell) \geqslant i$ as vertices $x_1, x_\ell$ already had edges incident of label $i$. Thus, in this case label of $e$ would be set to $i + 1$ (and not $i$) which contradicts our assumption. □




**Lemma 3** *For $i \geqslant 1$, two components in $T_i$ cannot be connected via a path lying entirely in graph $G - (T_1 \cup \cdots \cup T_{i-1})$.*

**Proof:** Let us suppose there is a path $P$ lying in graph $G - (T_1 \cup \cdots \cup T_{i-1})$ that connects two components, say $C_1, C_2$, in forest $T_i$. Without loss of generality assume first edge in $C_1$ was labeled before first edge in $C_2$.

Let $x$ be vertex lying in $P \cap C_2$, and $e$ be edge in $P$ incident to $x$. Let $(y, z)$ be first edge in $C_2$ for which label is set to $i$, and $t$ be the time-stamp immediately before changing label of $(y, z)$.

We now make a few observations:

- At time stamp $t$, $L(y) = L(z) = i - 1$. This is because at time $t$ both $y, z$ would have no edges of label $i$ incident to them, and so by Lemma 1, $L(y)$ and $L(z)$ must be $i - 1$.

- At time $t$, all edges in $C_1$ are labeled. This is because otherwise at time $t$, $C_1$ will contain an unlabeled edge whose one endpoint has label $\geqslant i$ and its priority (see step 4 of Algorithm) would thus be greater than $(y, z)$. Since this is not possible, all edges in $C_1$ must have been labeled at time $t$.

- No edges in $P$ can ever have labels in range $[1, i]$ as edges in $P$ do not lie in trees $T_1, \ldots, T_i$.

- At time stamp $t$, no edge in $P$ can have label 0 as otherwise the first such edge will have a higher priority then $(y, z)$ as one of its endpoint will have label at least $i$.

Now since $L(e) \geqslant i+1$, by Lemma 1, the component $C_2$ will already have an edge incident of label $i$ before labeling $(y, z)$ which violates the definition of $(y, z)$. So this contradicts the existence of $P$. $\square$

**An Efficient Implementation** In order to obtain an $O(m + n)$ time implementation we need a fast query procedure to report vertices in $V_0$ of highest label. This can be done as follows:

1. Compute an array $A$ of size $m + 1$. For each $i \geqslant 0$, we maintain in $A[i]$ a pointer to doubly-link-list $D_L[i]$ storing vertices $x \in V_0$ that satisfy $L[x] = i$.

   So in the beginning $L[0]$ stores $V$, and $L[1], \ldots, L[m]$ are empty.

2. Insertion and deletion of vertices in $D_L[i]$ can be done in $O(1)$ time as we can store pointer from vertices in $G$ to their position in doubly-link-lists $D_L[i]$'s.

3. In addition, we maintain a doubly-link-list $B$ that stores in decreasing order the indices $i \in [0, m]$ for which $D_L[i]$ is non-empty.

4. The updates in $B$ after changing label of an edge $e = (x, y)$ from 0 to a non-zero value can also be handled in $O(1)$ time as the label of vertices $x$ and $y$ are in worst case incremented by value at most 1.

5. Finally, note that updates in $B$ can also occur due to deletion of vertices from $V_0$. This can again be handled in $O(1)$ time as for each $x$ deleted from $V_0$, we can retrieve $i = L[x]$ and $|D_L[i]|$. In case $|D_L[i]|$ was 1, then $i$ needs to be deleted from $B$. Such a update can be done in $O(1)$ time as for each $i \in [0, m]$, we can store a pointer from $i$ to its position in $B$ (if $i$ lies in $B$).

Using doubly-link-list $B$ and lists $D_L[i]$'s, the vertex $x \in V_0$ of highest label can be reported in constant time.We thus obtain the following result.

**Theorem 1 (Nagamochi and Ibaraki (1992))** *For any multigraph $G = (V, E)$ with $n$ vertices and $m$ edges a $k$-edge connectivity preserver $H = (V, E_H \subseteq E)$ containing at most $nk$ edges can be computed in $O(m + n)$ time.*

**Remark.** Nagamochi and Ibaraki (1992) also proved that the subgraph $H$ obtained on taking union of edges of forests $T_1, \ldots, T_k$ computed by Algorithm 1 is a $k$-vertex-connectivity preserver.