Princeton Univ. F'24 COS 597b: Recent Advances in Graph Algorithms

Lecture 11: Decremental Strongly Connected Components and Single-Source Reachability in Near-Linear Time

Lecturer: *Ilya Maier, Ijay Narang, Baris Onat* Scribe: *Jessica Choe*

Reminder

We aim to maintain a hierarchy of separators as follows:

$$U = S_0 \supseteq S_1 \supseteq \cdots \supseteq S_{\log n+1} = \emptyset$$

where each strongly connected component (SCC) in $G_i = \text{Condensation}(G_{i-1} \cup E(S_i \setminus S_{i+1}))$ with $G_0 = (U, E(S_0 \setminus S_1))$ has "low" diameter (approximately $\tilde{O}(1)$) with respect to the S_i-distance. Here:

- S-distance is the number of nodes from S on the shortest path (SP) from u to v, or equivalently, setting weights of $E_{\text{out}}(S)$ to 1.
- *Condensation* means contracting SCCs into one vertex, while preserving loops and multi-edges.

1 Finding Good In-Out Separators

Input: A graph G, a root $r \in U$, a subset $U \supseteq S$, and a positive distance d. **Output:** $(U_{\text{reach}}, S_{\text{sep}})$ such that:

1. For all $v \in U_{\text{reach}}$, the distance from r to v (measured with respect to S) is at most d.
2. For all $x, y \in U_{\text{reach}} \cup S_{\text{sep}}$, $\text{dist}(r, x, S) = \infty$, meaning that x and y are unreachable.
3. The size of the separator satisfies:

$$|S_{\text{sep}}| \leq \min(|U_{\text{reach}} \cap S|, |U_{\text{unreach}} \cap S|) \cdot 2 \log n$$

4. The runtime for finding these sets is $O(E(U_{\text{unreach}}))$, where $E(U_{\text{unreach}})$ represents the edges of the unreached set.

Procedure: We place vertices into layers L_i:

- Assign the outgoing edges from S a weight of 1, and edges inside S a weight of 0.
- Start from r and run a breadth-first search (BFS) on the edges with weight 0, placing found vertices into layer L_0. For subsequent layers, use the outgoing edges with weight 1 to move from L_0 to L_1, then from L_1 to L_2, and so on.
- Look at $E_{\text{out}}(S_i)$ with weight 1 and add those neighbors to L_{i+1}.

- Contract all found vertices into r and continue.
- Goal: Find a "good" layer and set $S_{\text{sep}} = L_i \cap S$.

The reachable set is given by:

$$U_{\text{reach}} = \bigcup_{j \le i} L_j \cap S$$

Why does such an $i \le d$ exist?

Consider first point i' such that the sum of vertices captured from S in all layers up to i' exceeds half of the total size of S. This is formally expressed as:

$$\sum_{j \le i'} |L_j \cap S| > \frac{1}{2}|S|$$

In simpler terms, as we go through layers of the graph, we eventually capture more than half of the vertices from S by the time we reach layer i'.

Before reaching layer i', for every layer i (that is, for all $i < i'$), the number of vertices in S that we capture in each layer grows proportionally. Specifically, the number of vertices captured in each layer i is at least as large as the total number of vertices captured so far, divided by a constant d. For each $i < i'$, we have

$$|L_i \cap S| \ge \frac{\sum_{j \le i} |L_j \cap S|}{d}$$

This inequality suggests that as we move forward, the number of vertices in S being captured keeps increasing by a certain amount at each step.

By compounding this growth across layers, the total number of vertices from S captured up to layer i increases exponentially as the following:

$$\sum_{j \le i} |L_j \cap S| > \left(1 + \frac{2 \log n}{d}\right)^i$$

The factor $\left(1 + \frac{2 \log n}{d}\right)^i$ shows that the more layers we process, the faster we capture the vertices from S.

Case 1: If $i' > d/2$, then by applying the exponential growth formula with $i' = d/2 + 1$, we obtain:

$$\left(1 + \frac{2 \log n}{d}\right)^{d/2+1} > n$$

This leads to a contradiction, as it implies that we've captured more than the total number of vertices in the graph, which is at most n. Therefore, this case cannot occur, and i' must be less than or equal to $d/2$.

Case 2: If $i' \le d/2$, we apply a similar argument, but for the remaining vertices that haven't yet been captured. We handle this symmetrically:

$$\sum_{j \le i} |L_j \cap S| \text{ for the remaining vertices behaves similarly.}$$

This ensures that even in this case, we capture at least half of the vertices in S by the time we reach layer i'. Hence, there is no contradiction in this case, and the proof holds.

2 Split(G, S, d)

Input/Output

- The function takes in a graph G, a set of vertices S, and a distance parameter d.
- It outputs a tuple (S_{split}, P), where:
  - S_{split} is the set of separators (i.e., vertices that, when removed, divide the graph into disconnected components).
  - P is a partition of the vertices $U \setminus S_{\text{split}}$ into strongly connected components (SCCs).

Properties of the Output

- Each SCC in the partition P has an S-diameter less than or equal to d. The S-diameter refers to the maximum distance between vertices in the component when distances are measured according to the edges in S.
- For any pair of components x and y in the partition, there are no edges going from x to y or vice versa in the graph after removing S_{split}. In other words, x and y are fully disconnected from each other.
- The size of the separator set S_{split} is guaranteed to be relatively small, specifically bounded by $16 \log^2 n \cdot |S|/d$, ensuring that the separator set doesn't grow too large.
- The runtime of the algorithm is $\tilde{O}(md)$, where m is the number of edges in the graph.

Procedure

1. **Initialize Variables:** Start with an empty separator set S_{split} and an empty partition P.
2. **Pick an Arbitrary Vertex r and Run Two Parallel Separator Calculations:**

- Choose an arbitrary root vertex r from the set of vertices U.
- Run two algorithms in parallel to compute separators:
  - One for the *outgoing* separator (captures vertices reachable from r within a distance $d/8$ in the outgoing direction).
  - One for the *incoming* separator (captures vertices from which r can be reached within a distance $d/8$ in the incoming direction).
- The algorithm selects the result from whichever separator finishes first, storing the separator set and the vertices reachable by it as $(S_{\text{sep}}, U_{\text{reach}})$.
- If the number of reachable vertices $|U_{\text{reach}}|$ is less than or equal to two-thirds of the total number of vertices in V, proceed to the next step.
- Otherwise, wait for the other separator to finish and select its result.

3. **Case 1: If $|U_{\text{reach}}| \leq \frac{2}{3}|V|$:**

- If the number of reachable vertices is relatively small (less than two-thirds of the graph), we recursively apply the *Split* algorithm on the subgraph induced by U_{reach}. This ensures that we only work on smaller portions of the graph at a time.
- The result of this recursive call is a new separator set S'_{split} and a partition P' of the subgraph.
- We then:
  - Add the new separator set S'_{split} to our global separator set S_{split}.
  - Add the partition P' to our global partition P, and treat the separator S_{sep} as its own SCC (since it's separating the graph).
  - Update the graph G by removing the vertices in U_{reach} and S_{sep} from further consideration.

4. **Case 2: If $|U_{\textbf{reach}}| > \frac{2}{3}|V|$:**

- If the number of reachable vertices is larger than two-thirds, we initialize a generalized *ES-tree* (a specialized data structure for managing reachability information in the graph).
- The ES-tree is initialized with the root r, the set S, and the distance parameter d, but the distance threshold is reduced to $d/2$ to handle large portions of the graph more carefully.
- **Unreachable Vertices:** The ES-tree is used to identify any vertices that are unreachable from r. For each unreachable vertex v:
- If the distance between r and v exceeds $d/2$, we run the *InSeparator* algorithm from v, using a distance threshold of $d/4$.
- If not, we apply the *OutSeparator* process instead, looking for separators in the outgoing direction.
- In either case, we split the subgraph induced by U_{reach}, treating the unreachable vertices as their own components and adding this result to our overall output.

5. **Remove Processed Vertices:**

- After processing the unreachable vertices, we remove both the separator and the vertices it isolates (i.e., $U_{\text{reach}} \cup S_{\text{sep}}$) from the ES-tree and from the graph G.
- The graph is updated, and these vertices are no longer considered in the subsequent steps.

6. **Update the Partition P:**

- Once all reachable and unreachable vertices are processed, the remaining vertices in the ES-tree are collected, and the final partition P is updated with these vertices.
- The procedure ends with the final separator set S_{split} and the partition P, which are returned as the result.

Lemma

For any ϵ_r, we have:

$$|\epsilon_r.\text{GetAllVertices}| \geq \frac{1}{3}|V|$$

This means that at least one-third of the vertices in V will always be included in the current set handled by the GES-tree rooted at r.

Also, for the in/out reachable sets:

$$|V_{\text{reach}}^{\text{in/out}}| > \frac{2}{3}|V| \quad \text{with distance} \quad d/8$$

That is, we can always reach at least two-thirds of the vertices with respect to the given distance parameter $d/8$.

Now consider the ball of reachable vertices:

$$B_{\text{out/in}}(r) = \{v \mid \text{dist}(r, v, S) \leq d/8\}$$

Then:

$$|B_{\text{out}}(r) \cap B_{\text{in}}(r)| > \frac{1}{3}|V|$$

Thus, there exists an SCC with diameter $\leq d/4$ containing at least one-third of the vertices of V.

Proof Sketch: To show this formally, observe that the in/out reachable sets are constructed by running a breadth-first search (BFS) on G, expanding from the root r. The BFS layers correspond to the distances from r with respect to S-distance. Given the nature of BFS and how distances are computed, the sets $B_{\text{in}}(r)$ and $B_{\text{out}}(r)$ will each include at least two-thirds of the vertices before reaching a separator. The intersection of these sets forms an SCC with low diameter, bounded by $d/4$, due to the structure of the separator hierarchy.

The guarantees on the size of $B_{\text{in}}(r) \cap B_{\text{out}}(r)$ follow from the recursive application of the splitting algorithm, which ensures that large SCCs are broken down into smaller components. The proof relies on the fact that each step reduces the size of the reachable sets while preserving connectivity properties in the SCCs.

Consider the set S that is unreachable. If:

$$\text{dist}(r, s, S) > \frac{d}{2}$$

we compute the in-ball from S with radius $\leq d/4$. The number of vertices charged is:

$$|x \cap S| \cdot \frac{16 \log n}{d}$$

This ensures that the separator size is appropriately bounded by the logarithmic factor, maintaining efficiency.

Now, for the case where the in-ball satisfies $|B_{\text{in}}| \leq \frac{2}{3}|V|$, each time we remove the out-ball, we remove at most:

$$\frac{16 \log n}{d} \cdot |x \cap S|$$

3 Initialization

Let $\delta = 64 \log^2 n$.

1. Set $S_0 \leftarrow V$
2. Initialize the graph $G_0 \leftarrow (V, E)$
3. For $i = 0, \ldots, \lfloor \log n \rfloor + 1$:

$$\begin{aligned} (S_{\text{sep}}, P) &\leftarrow \text{Split}(G_i, S_i, \delta/2) \\ S_{i+1} &= S_{\text{sep}} \\ G_i &\leftarrow \text{Condensation}(G_{i-1}) \cup E(S_i \setminus S_{i+1}) \\ G_{i+1} &\leftarrow (P, E(S_i \setminus S_{i+1})) \end{aligned}$$

For each SCC $x \in P$, choose a root uniformly at random $r \in x$:

$$\epsilon_r \leftarrow \text{InitGES}(G_i[x], r, S_i, \delta)$$

The initialization process begins by setting the initial separator to be the entire vertex set $S_0 \leftarrow V$. We then recursively partition the graph using the split procedure, reducing the diameter of SCCs at each step. The condensation step contracts the SCCs while preserving key edges, ensuring that the structure of the graph is maintained. The GES-tree for each SCC is initialized using a uniformly selected root to ensure randomness in the algorithm.

4 Handling Deletions

To handle the deletion of an edge (u, v), we proceed as follows:

1. For $i = 0, \ldots, \lfloor \log n \rfloor + 1$, if there exists an SCC $x \in V_i$ such that both $u, v \in x$, call:

$$\epsilon_r.\text{Delete}(u, v)$$

2. While there exists an SCC x such that $x = \epsilon_r.\text{GetUnreachable} \neq \emptyset$:
   - If $\text{dist}(r, x, S) > \delta$, find an in-separator for r with $\delta/2$.
   - Similarly, find an out-separator for r with $\delta/2$.
3. If $|V_{\text{reach}}| \leq \frac{2}{3}|x|$, recurse on V_{reach}:
   - Split and initialize the GES for all newly found SCCs, and remove them from ϵ_r.
4. Otherwise, call:

$$\epsilon_r.\text{Delete}(x)$$

and split x:

- Repeat the same procedure as during initialization.

When an edge is deleted, we need to update the GES-tree to reflect this change. First, we identify if the edge (u, v) lies within the same SCC. If it does, we remove it using the GES-tree's delete operation. If this causes a vertex to become unreachable, we handle it by either finding separators to split the graph or by recomputing the GES-tree for smaller SCCs. This recursive approach ensures that the graph remains well-partitioned after the deletion.

Handling Level $i + 1$

In the else case, with probability $\geq 2/3$, each SCC output by the split has size $\leq \frac{2}{3}|x|$. Let $V_1, V_2, \ldots$ be the updates to ϵ_r. Define:

$$X^i_{\max}$$

as the SCC in X with diameter $\leq \delta/2$ and of maximum size after i updates.

This part of the algorithm deals with how to maintain the separator hierarchy as we handle updates. After a deletion, we may need to split SCCs into smaller parts. Each time, with high probability, we reduce the size of the SCCs by at least a factor of $2/3$, ensuring that the overall size of the graph decreases at each level of the hierarchy. By maintaining this structure, we ensure that the algorithm continues to operate efficiently, even as deletions are processed.

Each SCC is handled recursively, and after a sufficient number of updates, the SCC sizes shrink to a point where they no longer need to be split, maintaining the integrity of the GES-tree structure.