Lecturer: Anupam Gupta Scribe: Hui Han Chin, Jacob Imola

Dynamic graph algorithms is the study of standard graph algorithmic problems in the setup where the graph changes over time. For this lecture, it is assume that the vertex set of the underlying graph is fixed and the graph changes on the edge set with update operations `Insert(u,v)`and `Delete(u,v)`, u and v being vertices. This is known as the **edge arrival model**

1 Dynamic Connectivity

Two connectivity queries that we would like to support are `Connected(u,v)`, which ask if 2 vertices are in the same connected component and `Connected(G)`, which ask if the graph connected. While this is a basic question to ask and many results have been shown, it has not been completely resolved yet.
Lets consider 2 naive approach to dynamic connectivity for 2 vertices.

1. Track updates to the graph as an adjacency list and run DFS at every query. This would take $O(1)$ for update and $O(m + n)$ for query.
2. Track the vertices pairwise connectivity as a $n \times n$ matrix. Check an entry in the matrix during query. This would cost $O(n^2)$ for update since an edge can affect the connectivity between 2 $O(n)$ sized component, and $O(1)$ for query.

This illustrates a trade-off between update and query. Below are some known results for dynamic connectivity.

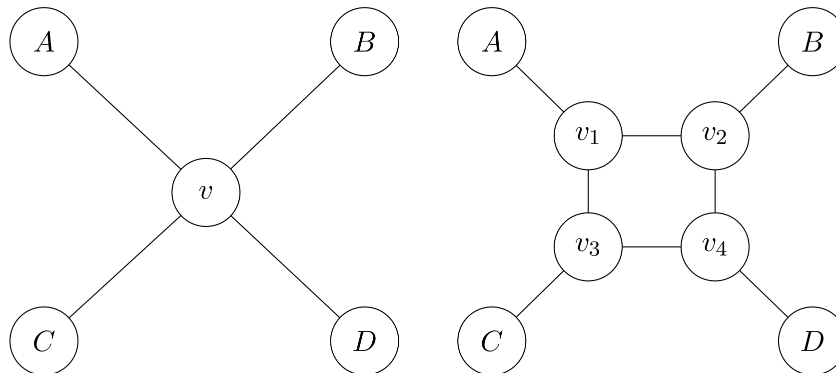
Authors	**Update Time**	**Query Time**	**Comments**
Frederickson [Fre85]	$O(m^{\frac{1}{2}})$	$O(1)$	Deterministic. Worst case
Eppstein et al [EGIN97]	$O(n^{\frac{1}{2}})$	$O(1)$	Sparsification
Holm, de Lichtenberg, Throp [HdLT98]	$O(\log^2 n)$	$O(\frac{\log n}{\log \log n})$	Amortized.
Kapron, King, Mountjoy [KKM13]	$O(\log^5 n)$	$O(\frac{\log n}{\log \log n})$	Randomized. Worst case

Let T_u and T_q be the update time and query time respectively. The known lower bounds, due to [PD04], are

$$T_u \times \log\left(\frac{T_q}{T_u}\right) \geq \log n$$
$$T_q \times \log\left(\frac{T_u}{T_q}\right) \geq \log n$$

2 Frederickson's Algorithm

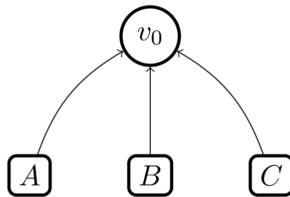
Below we provide two important ideas from Frederickson's algorithm [Fre85] that achieve $O(m^{\frac{2}{3}})$ update time. The first idea is pretending that every vertex in G has maximum degree 3. We can do this by treating any vertices v with degree $d > 3$ as d vertices connected to each other by a cycle where each one is connected to a distinct neighbor of v. The case for $d = 4$ is shown below.



This transformation does not change the connectivity properties of G. Thus, we will assume for the rest of this discussion that G has maximum degree 3.
The second idea is maintaining a clustered spanning forest F of G. We will cluster F in the following way:

Lemma 3.1. *Given any tree $T = (V, E)$ and a parameter z, where $|V| \geq z$, it is always possible to cluster V into partitions $V_1, V_2, \ldots, V_k$ such that each subgraph formed by V_i is connected and $z \leq |V_i| \leq 3z$.*

Proof. We will use induction on the size of V. Our base case is all trees of size at most $3z$; then, nothing needs to be done. Assuming $|V| > 3z$, orient each edge (a, b) in T in the following way: Remove (a, b) from the graph, and point to a if the tree with a in it is bigger than the tree with b in it. Otherwise, point to b. Since T is now an oriented tree, there must be a vertex v_0 which is a sink. We will analyze the case where v_0 has degree 3; the degree 1 and 2 cases are simpler. Suppose v_0 is connected to subtrees A, B, and C of sizes S_A, A_B, and S_C, as shown below:



Since $S_A + S_B + S_C + 1 > 3z$, not all three can be less than z. If all three are greater than z, then removing any edge connecting to v_0 will produce two subtrees on which we can apply the inductive hypothesis. So, without loss of generality, let $S_A < z$ and $S_B \geq z$. Since the edge e that goes between B and v_0 points to v_0, we must have $S_A + 1 + S_C \geq S_B \geq z$. Thus, cutting e will produce two trees which both have size at least z, so they are able to be clustered by the inductive hypothesis. The union of these clusters is a way of clustering T. Also, we can cluster T in $O(|E|)$ time using this recursive strategy. □

For each T in F we will maintain a set of clusters $C_1^{(T)}, C_2^{(T)}, \ldots, C_{s_T}^{(T)}$ if $|T| > z$ where each cluster has size between z and $3z$. If $|T| < z$, then just maintain one cluster around the entire tree. Each $C_i^{(T)}$ will keep track of all edges with at least one endpoint in $C_i^{(T)}$. Also, we will keep track of an adjacency list $A(T)$ which records which clusters in T have edges between them. Since the maximum degree is at most 3, there are $O(z)$ edges in total per cluster. The three operations are outlined below:

- `Insert(u,v)`: Let T_u be the tree containing u and T_v the tree containing v. If $u = v$, then we can add the edge to T_u and be done. So, suppose $u \neq v$. Let $C_i^{(T_u)}$ be the cluster containing u and $C_j^{(T_v)}$ be the cluster containing v. We will merge $C_i^{(T_u)}$ and $C_j^{(T_v)}$. Merging is simply a matter of looking through the edges and vertices of $C_i^{(T_u)}$ and $C_j^{(T_v)}$, and takes $O(z)$ time. If the resulting cluster is too big, we will split it in the way described in the lemma which takes $O(z)$ time. The total time spent is $O(z)$.

- `Delete(u,v)`: Let T be the tree containing (u, v). Let $C_i^{(T)}$ be the cluster containing u and $C_j^{(T)}$ be the cluster containing v. If $i \neq j$, then skip to the next paragraph. Assuming $i = j$, we will check if deleting this edge makes $C_i^{(T)}$ become disconnected. This takes $O(z)$ time. If it does, then we need to do more work; we will merge the two components of $C_i^{(T)}$ to any neighboring clusters if they are too small. If the resulting clusters are too big, we will split them in the way described in the lemma.
  Finally, we need to check if deleting (u, v) makes T become disconnected. We will do a graph search on $A(T)$ to see if T is disconnected, and if so, determine which clusters should be split off. This takes $O\left(\left(\frac{m}{z}\right)^2\right)$ time total.

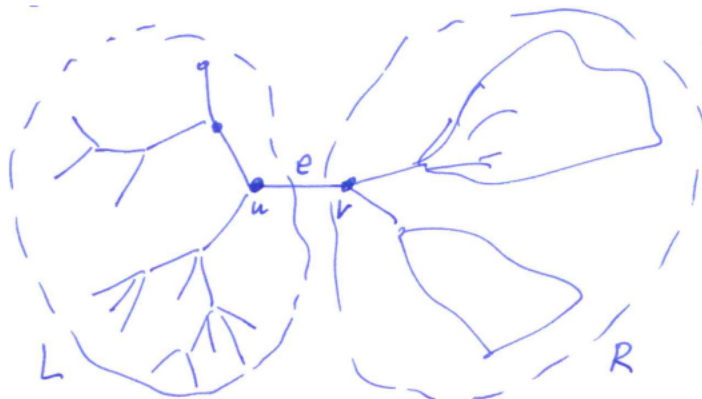
- `Query`: Since we have a spanning forest, queries are trivial to answer. We can do it in $O(1)$ time.

The runtime of `Insert` is $O(z)$ and the runtime of `Delete` is $O(z + (m/z)^2)$. Setting $z = m^{\frac{2}{3}}$ gives the best runtime for insertion and deletion of $O(m^{\frac{2}{3}})$.

3 Using Randomization in Dynamic Connectivity

Kapron-King-Mountjoy in SODA 2013 [KKM13] showed a dynamic connectivity algorithm that uses randomization. It has $O(\text{polylog}n)$ worst case update and $O(\text{polylog}n)$ query with 1-sided error, meaning "Yes" is always correct while "No" has probability $\frac{1}{n^c}$ error.
Similiar to most of the techniques so far, KKM13 relies on maintaining a spanning forest using dynamic trees. The main innovation is on identifying a candidate replacement edge on `Delete`. The high level ideas of their technique are as follows:

1. Consider a model where there are a series of `Insert` followed by just one `Delete`. This is needed to simplify the analysis for this lecture.
2. Each vertex,v, define a $O(\log n)$ bit label called $l(v)$
3. Each edge, $e = (u, v))$ define a label formed by the concatenation of the labels of its vertices $l(e) = \langle l(u)l(v)\rangle$ for some ordering of the vertices.
4. Let the bit XOR operator be $\bigoplus$.
   For a given vertex V, define $sign(v) = \bigoplus l(e)$, e are incident edges to x
   For a given subset of vertices S, define $sign(S) = \bigoplus_{v\in S} sign(v)$
   Notice that $sign(S)$ is $\bigoplus l(e)$, where e are edges which has exactly 1 endpoint in S.
5. The signature, $sign(v)$ operation can be done on dynamic tree data structure, such as the `Range` operation on link-cut trees, in $O(\log n)$



Suppose on `Delete(u,v)`, edge e is being remove. Let L be the subtree containing, u and R be the subtree containing v.
Consider candidate replacement edges between $L \to R$ in $G\backslash e$.

If there is no replacement edge, meaning $G\backslash e$ is disconnected, then $sign(L) = sign(R) = 0$
If there is only 1 replacement edge, f, between L to R, then $sign(L) = sign(R) = sign(f)$
If there are many possible replacement edges then $sign(L)$ or $sign(R)$ does not help in identifying the replacement edge.

The technique to find that replacement edge in the cutset, size greater than 1, is to sample. Suppose $O(\log n)$ sets is maintain for each vertex. Sample an edge to the i-th set with probability 2^{-i} while doing `Insert`. For this to work, one of these sets needs to have exactly one unique edge. Notice that if the cut set has C edges, then the probability of the $\log C$ th set having exactly 1 edge is:

$$C\frac{1}{2^i}(1-\frac{1}{2^i})^{C-1} = C\frac{1}{2^{\log C}}(1-\frac{1}{2^{\log C}})^{C-1} = (1-\frac{1}{C})^{C-1} > \frac{1}{\mathrm{e}} \in O(1)$$

Hence, with high probability of $1 - \frac{1}{\text{polylog}n}$, there will be some set with exactly 1 edge, thus the scheme works.

The first `Delete` on the tree is depends only on the state of the tree when delete occurs. However, subsequent `Delete`s would have dependences on other `Delete`. Thus the calculations in the above analysis cannot use indepedence the same way. Refer to [KKM13] for the technique to get around this limitation in the analysis.

4 Amortized analysis for $O(\text{polylog}n)$ deterministic bound

We would present a high level idea of [HdLT98] to a show dynamic connectivity algorithm with $O(\log^2 n)$ amortized bound.
Consider storing the graph G as $\log n$ subgraph, each with a minimum spanning forest. For each edge, have a "level" that is an integer in $[0, \log n]$. This would be the potential function used for the amortized analysis. An edge is inserted at level 0 and moves up 1 level everything it is scanned during a delete operation.
Let G_i be the subgraph of G consisting of the edges of level i and F_i be its spanning forest. Each of the forests, F_i, is maintained by a dynamic tree such as Link-Cut Trees where the amortized cost for each of these tree operations is $O(\log n)$.
2 invariants would be maintained throughout the execution of the algorithm.

1. I1: Every connected component of F_i has at most 2^i vertices.
2. I2: $F_{\log n} \subseteq F_{\log n-1} \cdots \subseteq F_0$. Also this means if $u, v \in G_i$ are connected, then u, v are connected in F_i

On `Insert(u,v)`, add the edge to G_0 and update the associated data structures in $O(\log n)$.

On `Delete(u,v)`$=e$, if edge, e is at level l, look at the tree, T which contains e. Split T into sub tree L, R such that $|L| \leq |R|$ and raise the edges of L to level $l+1$. This maintains I1 as $|L| \leq \frac{|T|}{2} \leq \frac{n}{2^l} * \frac{1}{2}$ by induction.

Now start scanning the *non-tree* edges incident to L at level l.
If the edge is within L, raise it to level $l+1$.
If it goes to R, then we have a replacement edge, and add it to $F_l, F_{l-1}, \cdots, F_0$ to maintain I2.
Suppose that there no replacement edges at level l, then rerun the process at level $l-1$.
Each edge charged $O(\log n)$ times thus it cost $O(\log^2 n)$ per `Update`.

On `Connected(u,v)`, check if both u, v are in the same component by I2. This can be done in $O(\log n)$ time in the dynamic tree.

5 Key Ideas for the Lecture

1. Clustering and tree separators as seen in Fredrickson.
2. Amortized analysis by Holm et al.
3. Bit tricks such XOR to detect cuts and use of sampling as seen in KKM13.

References

[EGIN97] David Eppstein, Zvi Galil, Giuseppe F. Italiano, and Amnon Nissenzweig. Sparsification;a technique for speeding up dynamic graph algorithms. *J. ACM*, 44(5):669–696, September 1997. 1

[Fre85] Greg N. Frederickson. Data structures for on-line updating of minimum spanning trees, with applications. *SIAM Journal on Computing*, 14(4):781–798, 1985. 1, 2

[HdLT98] Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, pages 79–89, New York, NY, USA, 1998. ACM. 1, 4

[KKM13] Bruce M. Kapron, Valerie King, and Ben Mountjoy. Dynamic graph connectivity in polylogarithmic worst case time. In *Proceedings of the Twenty-fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '13, pages 1131–1142, Philadelphia, PA, USA, 2013. Society for Industrial and Applied Mathematics. 1, 3, 3

[PD04] Mihai Pătraşcu and Erik D. Demaine. Lower bounds for dynamic connectivity. In *Proceedings of the Thirty-sixth Annual ACM Symposium on Theory of Computing*, STOC '04, pages 546–553, New York, NY, USA, 2004. ACM. 1